1989-12

Application of VAX/VMS graphics for solving preliminary ship design problems

McGowan, Gerald K.

Monterey, California : Naval Postgraduate School

(2)

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

APPLICATION OF VAX/VMS GRAPHICS
FOR SOLVING PRELIMINARY
SHIP DESIGN PROBLEMS

by

Gerald K. McGowan

March, 1990

Thesis Advisor: Fotis A. Papoulias

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (*If applicable*) 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (*If applicable*)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS Program Element No. / Project No. / Task No. / Work Unit Accession Number
11 TITLE (*Include Security Classification*) APPLICATION OF VAX/VMS GRAPHICS FOR SOLVING PRELIMINARY SHIP DESIGN PROBLEMS	
12 PERSONAL AUTHOR(S) McGowan, Gerald Keith	
13a TYPE OF REPORT MASTER'S THESIS 13b TIME COVERED From To	14 DATE OF REPORT (*year, month, day*) 1990, MARCH, 29 15 PAGE COUNT 129
16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.	
17 COSATI CODES FIELD / GROUP / SUBGROUP	18 SUBJECT TERMS (*continue on reverse if necessary and identify by block number*) VAX/VMS GRAPHICS, VAX/VMS PROGRAMMING, PRELIMINARY SHIP DESIGN, STATIC STABILITY, HYDRODYNAMIC COEFFICIENTS, SHIP DESIGN.

19 ABSTRACT (*continue on reverse if necessary and identify by block number*)

The VAX/VMS UIS graphics library routines were used in the creation of a menu driven, interactive program which solves basic preliminary ship design problems. The program uses a menu with active mouse and keyboard to select options, enter data, and control program execution.

At present, the program solves transverse and longitudinal static stabilty problems and predicts the effects of shifting weights in three planes. It also calculates the hydrodynamic derivatives for maneuvering performance and predicts the turning circle characteristics of the ship. Provisions for a hardcopy, detailed report are also included. Space has been allocated to include future program modules or user supplied programs.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Fotis A. Papoulias	22b TELEPHONE (*Include Area code*) (408) 646-3381 22c OFFICE SYMBOL Code 69PA

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

Approved for public release; distribution is unlimited.

Application of VAX/VMS Graphics
for Solving Preliminary
Ship Design Problems

by

Gerald K. McGowan
Lieutenant, United States Navy
B.E.E., University of Washington

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

Author: 
Gerald K. McGowan

Approved by: 
Fotis A. Papoulias, Thesis Advisor


James F. Hallock, Second Reader


Anthony J. Healy, Chairman
Department of Mechanical Engineering

ABSTRACT

The VAX/VMS UIS graphics library routines were used in the creation of a menu driven, interactive program which solves basic preliminary ship design problems. The program uses a menu with active mouse and keyboard to select options, enter data, and control program execution.

At present, the program solves transverse and longitudinal static stability problems and predicts the effects of shifting weight in three planes. It also calculates the hydrodynamic derivatives for maneuvering performance and predicts the turning circle characteristics of the ship. Provisions for a hardcopy, detailed report are also included. Space has been allocated to include future program modules or user supplied programs.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

A valuable source of information, not mentioned in the References, was the COMPUSERVE VAX/VMS Forum. Specifically, the System Operators, Bill Mayhew and Stuart Fuller, provided information and guidance which was always accurate and very timely.

I. INTRODUCTION

A. PURPOSE

Modern programming techniques endeavor to make the program transparent to the user. This allows users, with limited computer experience, to concentrate on problem solution and not on computer operation. This project was undertaken to lay the groundwork for an in house program containing a variety of routines for solving preliminary ship design problems. It is intended for use by the faculty and students in various ship design courses offered in the Mechanical Engineering Department.

The program "Tool Box" also explores the application of MicroVMS graphics capabilities to the solution of engineering problems. It was designed to be a modular, interactive, user friendly, preliminary ship design tool. VAX FORTRAN was combined with User Interface Services (UIS) graphics routines to produce a menu driven, interactive program which can be used with a minimum of computer training.

B. BACKGROUND

FORTRAN has long been a favorite, high level, problem solving language. One major drawback in FORTRAN programming has always been its poor to non-existent graphics capabilities.

The MicroVMS is supported by a large library of User Interface Services (UIS) graphics routines. The UIS routines can be called from high level languages

such as VAX FORTRAN, VAX PASCAL, VAX C and VAX PL/I, as well as VAX MACRO and VAX BLISS.

The UIS graphics routines allow the programmer to create, delete and modify multiple display windows. The graphics routines also allow object and text drawing, using many different fonts, line types, colors and shading.

One very useful category, Asynchronous System Trap (AST) UIS routines, allow the programmer to enable routines which are triggered by specific run time events.

By combining the problem solving capabilities of VAX FORTRAN with the UIS graphics routines, user friendly programs can be written which solve complex problems and produce excellent graphics as well.

Recently, a project was started in the Naval Postgraduate School Aeronautics Department which utilized the graphics capabilities in a FORTRAN program for creating curves with point inputs from the mouse. Professor R.E. Ball was kind enouph to furnish a copy of the AE 4900 course report and program code. This proved to be an excellent starting point for the current project. [Ref. 1]

C. DOCUMENT STRUCTURE

The remaining chapters are arranged in an order of increasing detail.

Chapter II, OPERATING TOOL BOX, details the program capabilities and limitations. It then presents an example which takes the user through a problem step by step. This is as far as an operator needs to read to effectively use Tool Box.

Chapter III, TOOL BOX PROGRAM DESCRIPTION, describes in detail the program and subroutines which make up Tool box.

CHAPTER IV, VAX VMS GRAPHICS ROUTINES, is a detailed look at the UIS routines used in Tool Box. This is intended to provide a starting point for basic graphics programming.

D. CONVENTIONS USED

Whenever a specific keyboard key is referred to in the text, it will be all capitals enclosed in square brackets, [RETURN].

Program or subsection options will be in all capitals, STATIC STABILITY.

The calling statements will be in all capitals when referenced. This includes the arguments of the calling routine, CALL UIS$TEXT(VD_ID,4,2.5,3.0).

II. OPERATING TOOL BOX

A. INTRODUCTION

1. Program Capabilities

"Tool Box" is a menu driven, interactive program which performs several functions. The current program functions can be put into two broad categories of Static Stability and Maneuvering Performance.

2. Static Stability

The initial inputs in the Static Stability section are Ship Name, Length (Lpp and Lwl), Beam (B), Draft (design (T) and Trial (Tt)), Displacement or Block Coefficient (Cb), Prismatic Coefficient (Cp), Midships Coefficient (Cm), Vertical Prismatic Coefficient (Cvp), and Longitudinal Center of Gravity (Xg). Initial data is entered using the keyboard or by reading in a previously stored data file.

After data is entered or at any time an item is changed, the program calculates the Maximum Cross Sectional area (Ax), Waterplane area (Aw), Midships area (Am), Volume of Displacement (cubic feet), Waterplane Area Coefficient (Cwp), Length to Beam Ratio (L/B), Length to Draft Ratio (L/T), and an estimation of Vertical Center of Buoyancy (KB) using relationships derived from Introduction to Naval Architecture [Ref. 2].

a. Transverse Stability

INITIAL TRANSVERSE STABILITY has two sections. When the user furnishes or changes KG, GM, KM, BM or Transverse Stability Index; the

other variables are calculated and displayed. In order to predict specific ship values, two of the above items must be supplied.

Transverse Stability Index (TSI), is a term coined in this program to refer to the non-dimensional quantity of GM/KM. TSI is used as a design anchor in an attempt to reduce the number of required arguments for transverse stability. A reasonable value of TSI (say 0.18) plus a desired GM or KM, yields the remainder of the variables.

It would appear that TSI could be a useful quantity in that ships of similar type appear to have similar TSI. It also gives a very nice working range for the relative transverse stability of the ship, since it should be between 0 (borderline transverse stability) and 1.0 (maximum stability). This needs more study as the number of ships looked at is small. Several ships are tabulated and their calculated TSI are shown in Table 1.

After the ship's KG, GM, KM, and BM are calculated, the next section can be used to see the effects of weight addition or removal in the vertical and transverse directions. The effects on ships displacement, draft, list angle, KG, KM, GM, and BM are shown. The actual baseline ship data is not altered. In this way, many off-baseline effects can be studied without affecting baseline data.

SHIP	DISPLACEMENT	GM	KM	TSI
Mariner Standard Cargo	22630	5.0	31.09	.161
Break Bulk Cargo	21235	3.6	34.6	.104
Combination Passenger, reefer, Container	19799	3.2	33.8	.095
Container Ship	33924	5.0	40.15	.125
Barge Carrying (Gantry)	32800	8.0	48.82	.164
Barge Carrying (elevator)	44500	7.6	59.35	.128
Tanker	90,400	21	52.67	.399
Ore Carrier	66,200	15.7	42.2	.372
Survey Ship Discoverer	3702	3.12	24.0	.13
DD692	3000	3.7	24.0	.192

All data with the exception of DD692 are from Amelio M. D'Arcangelo, Ship Design and Construction, The Society of Naval Architects and Marine Engineers, p. 30, 1969. Data for DD692 is taken from [Ref. 1].

Table I. Transverse Stability Index for Several Ships

b. Longitudinal Stability

The Longitudinal Stability section is partitioned the same way as the Transverse section. Part 1 calculates GML, BML, KML, KG, Il, Cwl and moment to trim an inch whenever one of these values is entered or changed. Part 2 shows the effects of weight addition or removal along the longitudinal axis on displacement, trim, and trim angle. As in the transverse section, the weight shift changes do not alter the baseline data.

3. Maneuvering Performance

The University of Michigan has developed a program for use on the MACKINTOSH, which provides the user with a tool for estimating the hydrodynamic derivatives of a surface ship, based upon physical ship parameters. The program also predicts the turning and steering perfomance of the ship based upon those same parameters. This work provided a starting point for the capabilities to be programmed for Tool Box. It also provided a good sample problem to compare the results of the present project with. [Ref. 3]

The Maneuvering Performance section uses input data which is similar to that used for the static section, but the two are separate independent programs and data files should not be mixed.

Data for a baseline ship is input by keyboard or from a previously stored data file. The input consists of seventeen pieces of ship data as shown in Figure 1.

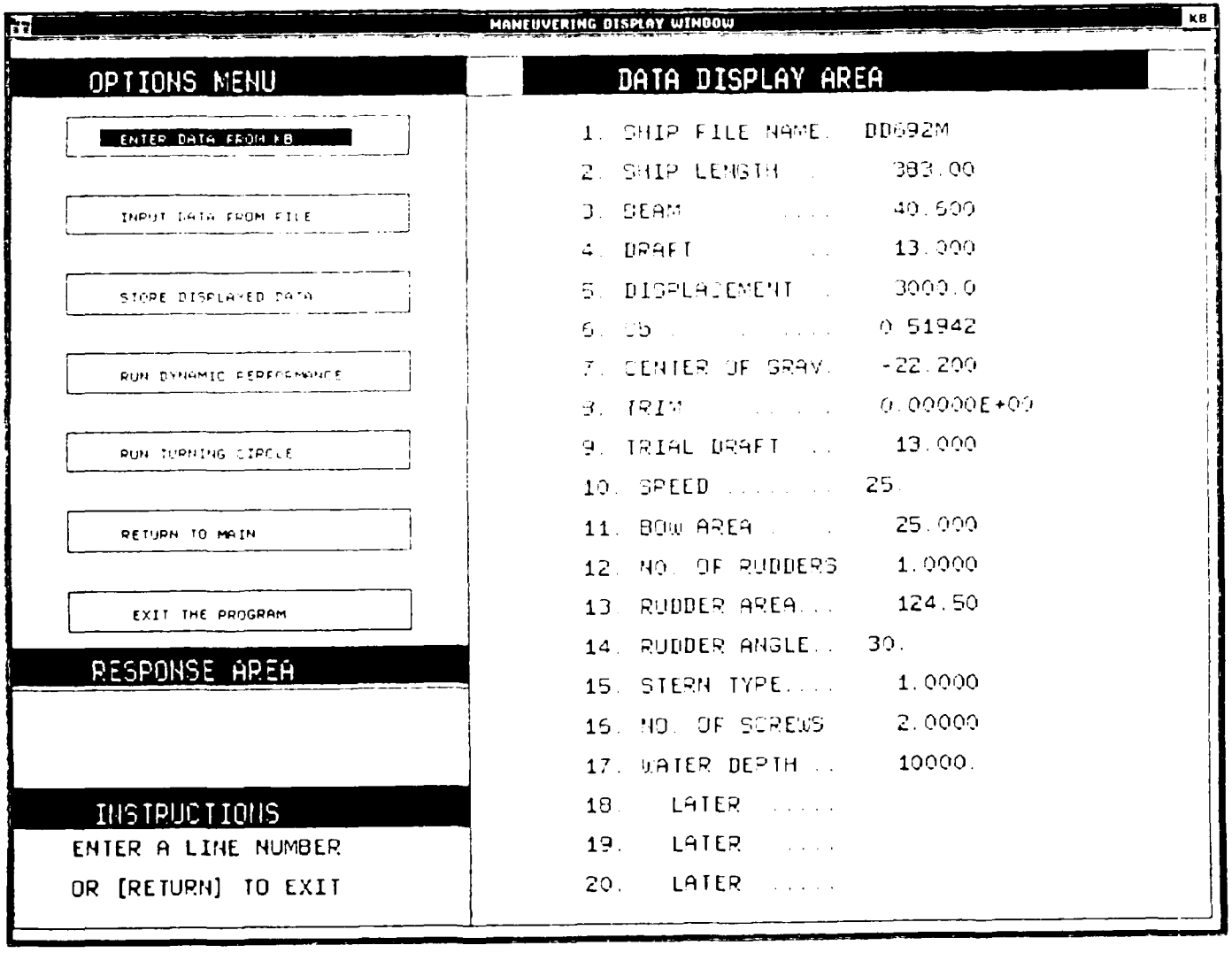

Figure 1 Maneuvering Display Window After Data Entry

After data input, either the Dynamic Performance or Turning Circle Performance can be calculated and displayed. The display data can also be stored at any point while in the Maneuvering section.

a. Dynamic Performance

The Dynamic Performance section uses the base line data to estimate the hydrodynamic coefficients. Then, based upon the coefficients, it calculates the turning index, stability index, and the dynamic stability of the ship. For a review of the derivation of the hydrodynamic coefficients and the estimating methods see Appendix A.

If the user has good estimates of the actual hydrodynamic coefficients, they can be input and the results of such a change on the performance is immediately displayed on the screen. When user supplied coefficients have been entered, they are also stored with the baseline data whenever the file is stored. These user supplied coefficients become part of the written report, if a report is generated.

b. Turning Circle Performance

The Turning Circle Performance can also be calculated from the input baseline data. Ship Length, Beam, Draft, Displacement, Block Coefficient, Rudder Angle and Approach Speed are displayed and can be changed. The changes cause the baseline data to be modified and are reflected in recalculation and display of Steady Turning Diameter (STD), Tactical Diameter (TD), Advance (AD), Transfer (TR), and Turning Speed (Vt).

Ship File Name is displayed in order to change the file name if the changes are to be saved in a different file.

4. Reports

The program will generate reports of either Static Stability or Maneuvering Performance if a data file has been saved from the section.

The Static Stability Report reads in the baseline data, does the section calculations and produces a user named file. The report file is formatted for a 59 line printer and provides a more complete set of calculations than the screen presentation. An example report is included in Appendix B.

A feature of the Static Stability Report section is the ability to perform several iterations of stability calculations over a specified range of TSI holding GM constant. This is not available in the interactive screen section.

The Maneuvering Performance Report is also generated from a previously stored baseline file. If the baseline file contains user supplied hydrodynamic coefficients, the report generates an additional two pages to incorporate the data. This report is also more detailed than the screen presentations and is formatted for a 59 line printer. An example of the Maneuvering Report is included in Appendix C.

5. Utilities

The FILE UTILITIES section allows the user to execute routine DCL commands without exiting the program and losing program time and data. Nearly all DCL commands can be run with the exception of changing the default directory. This will not work since the active directory is locked with Tool Box execution. Utilities can rename, move, edit and print files just to name a few DCL options.

The option is selected by moving the mouse pointer into the FILE UTILITIES rectangle and depressing the left mouse button. After the DCL commands have been run, return the Main Menu by depressing the [RETURN] key without data entry.

B. PROGRAM LIMITATIONS AND ASSUMPTIONS.

There are several limitations placed upon the program. Some of the limitations are software limitations and some are associated with the computation methods used.

Small angle approximations were assumed in the static stability calculations which limit the usefulness of the angles to less than seven degrees. In the weight shift calculations, it was further assumed that the water plane area and the moments of inertia were constant throughout the range of interest.

In Static Stability and Maneuvering Performance, the input character strings for ship/file name are limited to eight characters. When a file is stored, the ship/file name becomes the FILENAME. An automatic file extension of .DAT is used for stored data.

File names used in the Report section may contain 12 characters including a period. Generally the format is an eight character file name plus a period, plus a three character file extension (FILENAME.FILETYPE). If a longer name is to be given to a file, the utility section of the program can be used to RENAME the file with up to 64 characters. This 64 characters include filename and extension.

Input strings for numerical data can contain up to 12 characters including a period. All floating point information is handled in a G12.5 format.

The program is specifically written for salt water using English units of feet, long tons and inches. All angles are in degrees.

Files from Static Stability and Maneuvering Performance are not interchangeable and care must be used to keep from mixing the two.

C. AN EXAMPLE

In order to demonstrate Tool Box more fully, an example is presented. This example will use DD692 and applicable data as found in Reference 2.

1. General User Instructions

Tool Box was written to be easy to operate and there are only a few general rules for getting around the program.

To select an option, move the mouse pointer to the option rectangle. When the option highlights, depress the left button of the mouse. This initiates the option and instructions or status reports will be displayed.

If the program is executing a keyboard interactive subsection, the required action is presented as a user instruction or prompt.

In order to exit any keyboard interactive subsection, depress [RETURN] without any other data input and watch for new instructions. This terminates the option action and returns control to the displayed menu.

If EXIT THE PROGRAM is selected before data is stored, the data is lost.

Data which has been input remains active in its particular section/subsection until changed or until the main program is exited. The user is free to move around within the program without loss of data in any subsection.

2. General Observations.

Occasionally, the program will be slow to respond or will appear inactive for a short period of time. Simply wait a few seconds for the system to become responsive and continue the program.

It has been observed that the default display used in the Utilities mode sometimes does not automatically gain control of the keyboard when Utilities is called up. In these cases, place the pointer on the display and depress the left button to assign the keyboard.

Response time is increased if there is only one active window on VAX/VMS prior to running Tool Box.

3. Static Stability of DD692

a. Getting Started

Before starting Tool Box, assemble the data needed to run the problem. A worksheet for Static Stability is found in Appendix D. Log into the Micro Vax system as normal and RUN TB. A display screen, Figure 2, will be created on the screen.

Select STATIC STABILITY by moving the mouse pointer into the rectangle labeled STATIC STABILITY. When the rectangle becomes highlighted, depress the left button on the mouse.

Figure 2 Tool Box Main Menu Display

After a moment, the INITIAL STABILITY WINDOW, Figure 3, should appear as the active or top display.

b. Entering and Storing Data

The next step will be to create a data file from the keyboard. Place the mouse pointer on the ENTER DATA FROM KB option and when highlighted, depress the left mouse button. The display should change to resemble Figure 4, except, at this point, the data displayed is dummy data to prevent divide by zero errors during data entry.

At the bottom of the display, in the instruction area, note the instruction to ENTER A LINE NUMBER OR [RETURN] TO EXIT. This is where all instructions will be displayed or status written.

First, enter line number one (1), and [RETURN]. The instruction changes to ENTER A NEW SHIP NAME. Enter a ship name, DD692S, of up to eight characters. Notice that the keystrokes are echoed in the RESPONSE AREA of the display. Before [RETURN] is depressed, the line can be edited using backspace and retyping. When [RETURN] is pressed, the response area is cleared and the new ship name is displayed in its proper place on the DATA DISPLAY AREA.

Each item on the screen with a line number can be changed with the keyboard. Items without line numbers are calculated values only and can not be changed by keyboard entry.

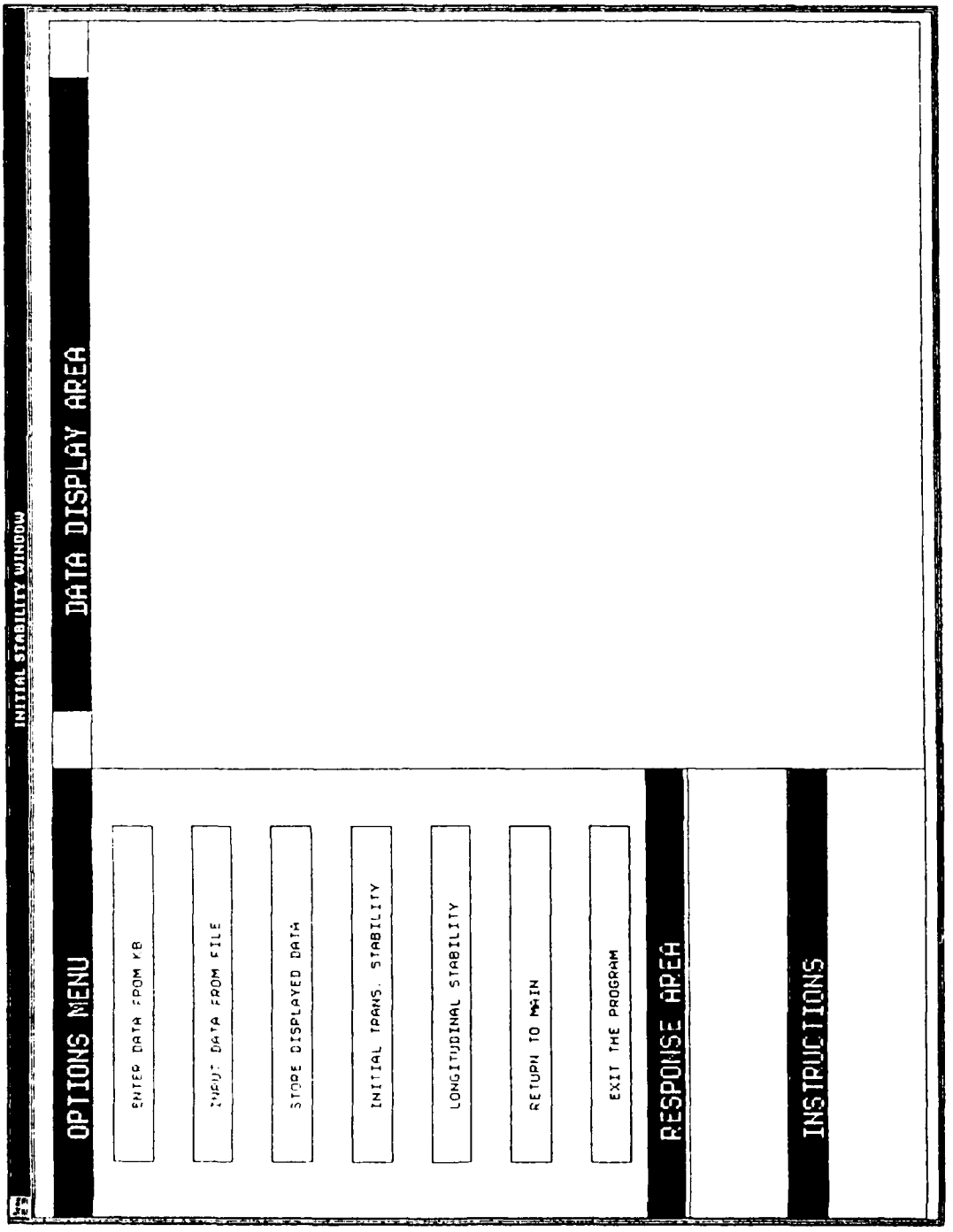

Figure 3 Initial Stability Window After Option Selection

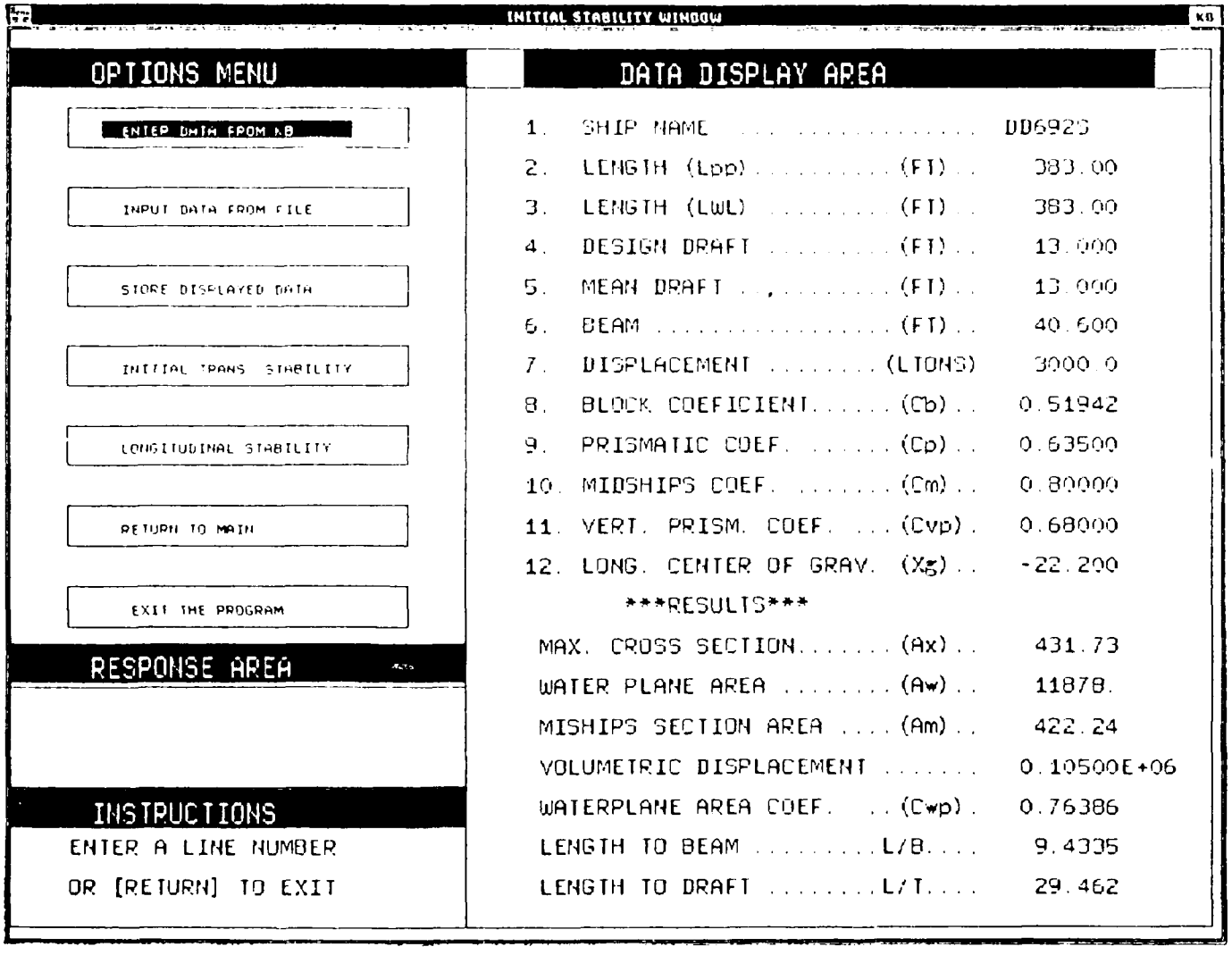

Figure 4 Initial Stability Window After Keyboard Data Entry

Enter the data as shown in figure 4. As data is entered, the ***RESULTS*** section is continually updated with each [RETURN] key.

When all data has been entered, depress the [RETURN] key one last time without any preceding data. The program returns control to the Initial Stability menu and the instruction will read SELECT AN OPTION WITH THE MOUSE.

At this point, select the STORE DISPLAYED DATA option. The input data is stored automatically, using the Ship Name as the filename and DAT as the file extension. The display should now look like Figure 5. This file DD692S.DAT, now has all of the data needed to reconstruct the data on the display, if needed in the future.

In order to read a previously stored file, select INPUT DATA FROM A FILE with the mouse. The instruction will be ENTER THE FILE NAME AND FILE EXTENSION. Key in the file name which contains the required data and enter [RETURN]. The contents of the file will be read in and the display will be like Figure 4.

It is important to remember that the files for Static Stability and Maneuvering Performance are formatted differently and should not be interchanged.

c. Initial Transverse Stability

After data has been entered, the program can begin stability calculations. Select the INITIAL TRANS. STABILITY option with the mouse. The display will change and will appear like Figure 6.

Figure 5 Initial Stability Window After Storing Data

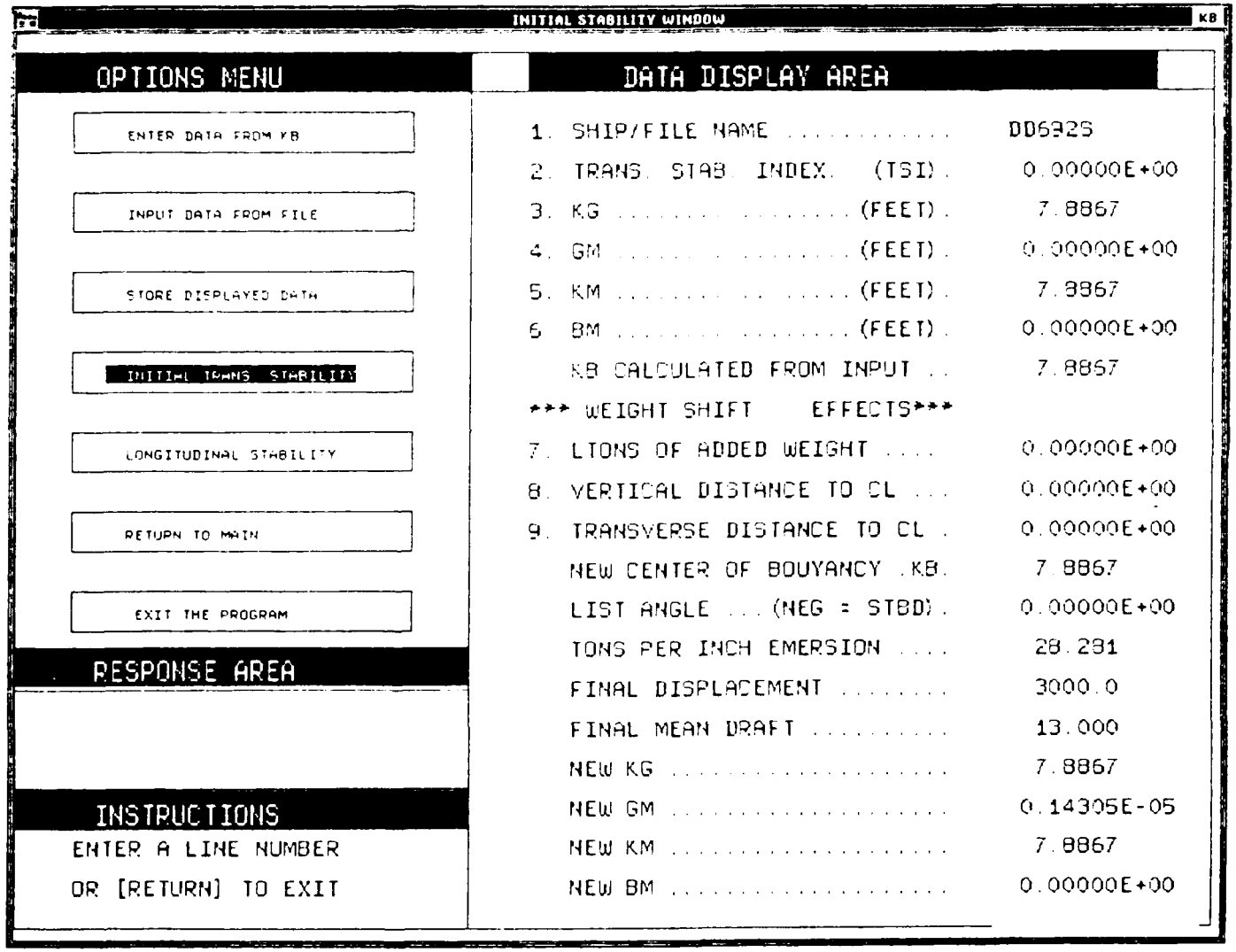

Figure 6 Transverse Stability Display

In order to get proper results, 2 of the data lines 2 through 6 must be supplied. At this time, enter a TSI of .192 and a GM of 3.7 feet. Notice that whenever an item is input, the others are recalculated and displayed at the [RETURN]. The display now appears like Figure 7.

After lines two to six have been computed, the weight shift portion of the program will give accurate predictions of the effects of weight change on the base line data. These calculations will not change the base line data. They serve only to show the predicted effects.

Enter a weight of 25 long tons at 2.5 feet above (positive) the current center of gravity and 3.2 feet to port (port = positive). The results will appear as in Figure 8.

Exit this option by depressing [RETURN]. The data just entered is stored by selecting STORE DISPLAYED DATA. This data file will have the same name as the previous file stored but it will now contain the Transverse Stability information. If, at some future time, this file is read into the program, all of the information, including the last entered weight shift effects will be displayed accordingly.

The user is free to select either ENTER DATA FROM KB or INITIAL TRANS. STABILITY at any time to alter any numbered line data and the results will be displayed.

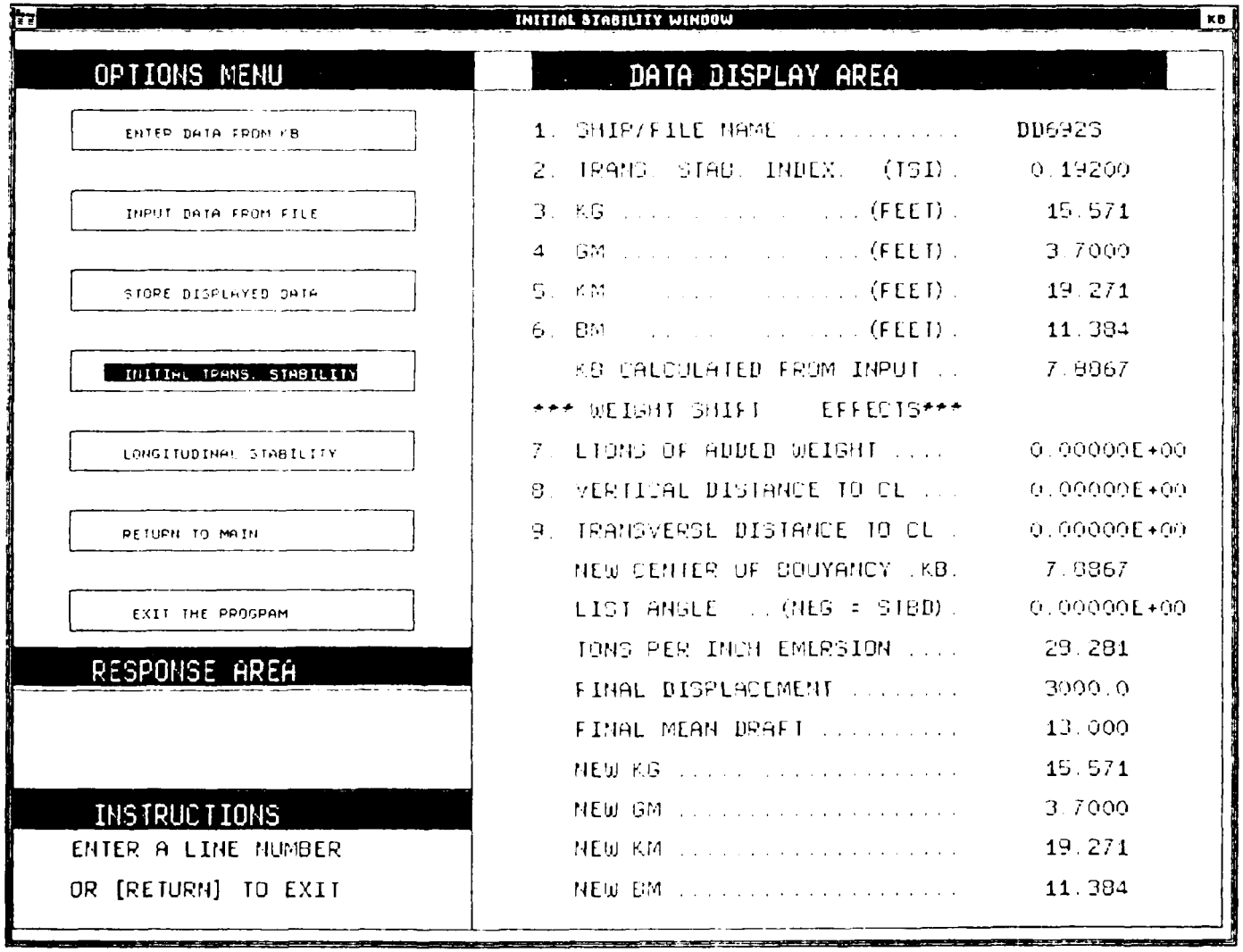

Figure 7 Transverse Stability With GM and TSI Entered

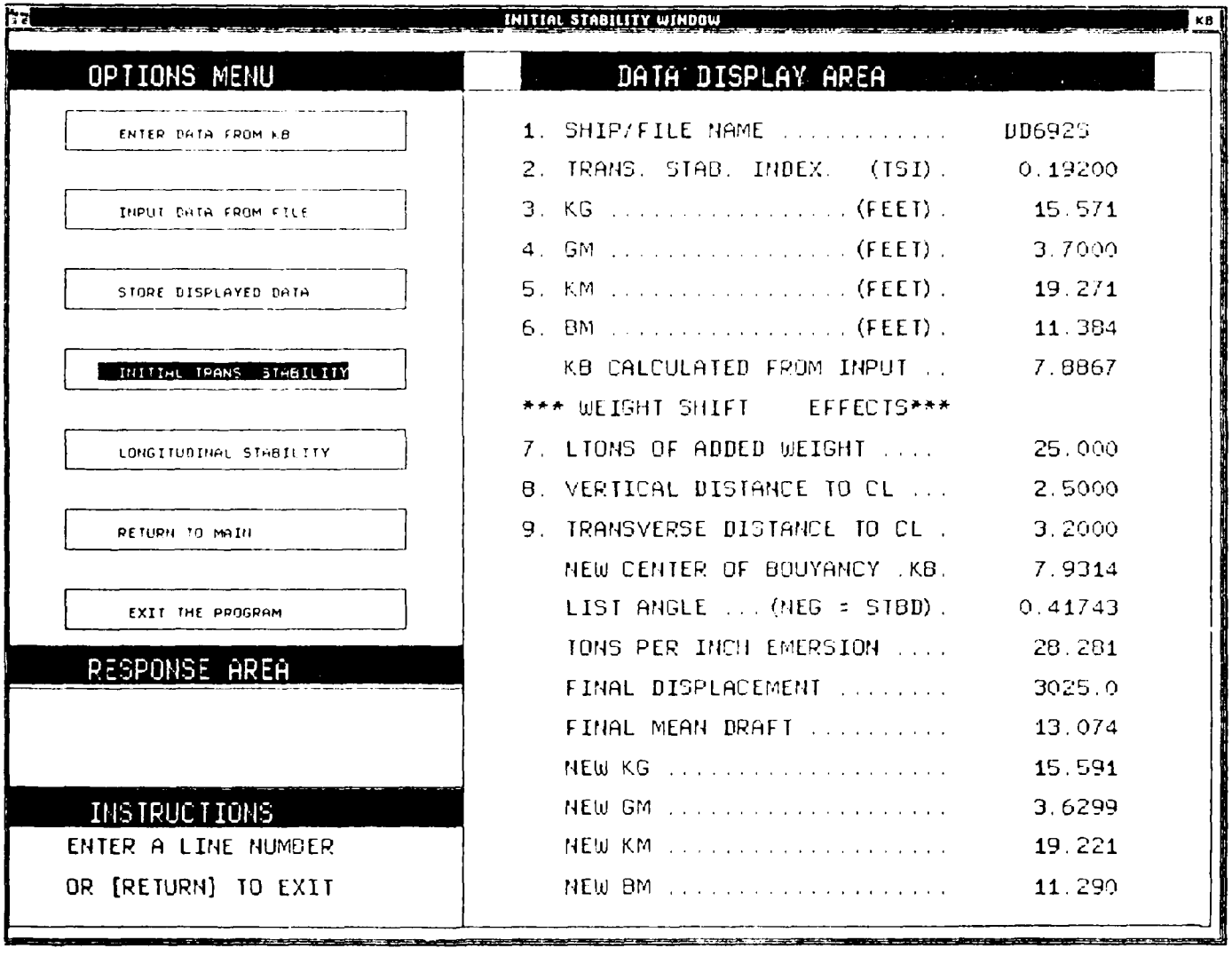

Figure 8 Effects of Weight Shift on Transverse Stability

d. Longitudinal Stability

The longitudinal stability section will perform longitudinal calculations after the base line data has been entered and the Initial Transverse Stability has been run.

Select the LONGITUDINAL STABILITY option with the mouse. One piece of data from lines two through eight must be supplied in order to calculate the others and enable longitudinal weight shift calculations. For this example, enter Moment to Trim an Inch of 650. The display should now appear as Figure 9.

Next, enter a weight of 22 long tons, 14 feet forward (positive) of midships. The display now appears as Figure 10.

Exit the Longitudinal Stability option and store the data file. This file DD692S.DAT now contains all of the information needed to recreate all three Static Stability screens and generate a more detailed written report.

Select the RETURN TO MAIN option to return program control to the MAIN MENU.

4. Maneuvering

Select the Maneuvering option of the Main Menu with the mouse. The screen should be changed to that of Figure 11.

Since the Maneuvering section uses a different data file than the Static section, the first step is to enter data from the keyboard unless a previously stored file is available. Select the ENTER DATA FROM KB option with the mouse. The screen will now look like Figure 12.

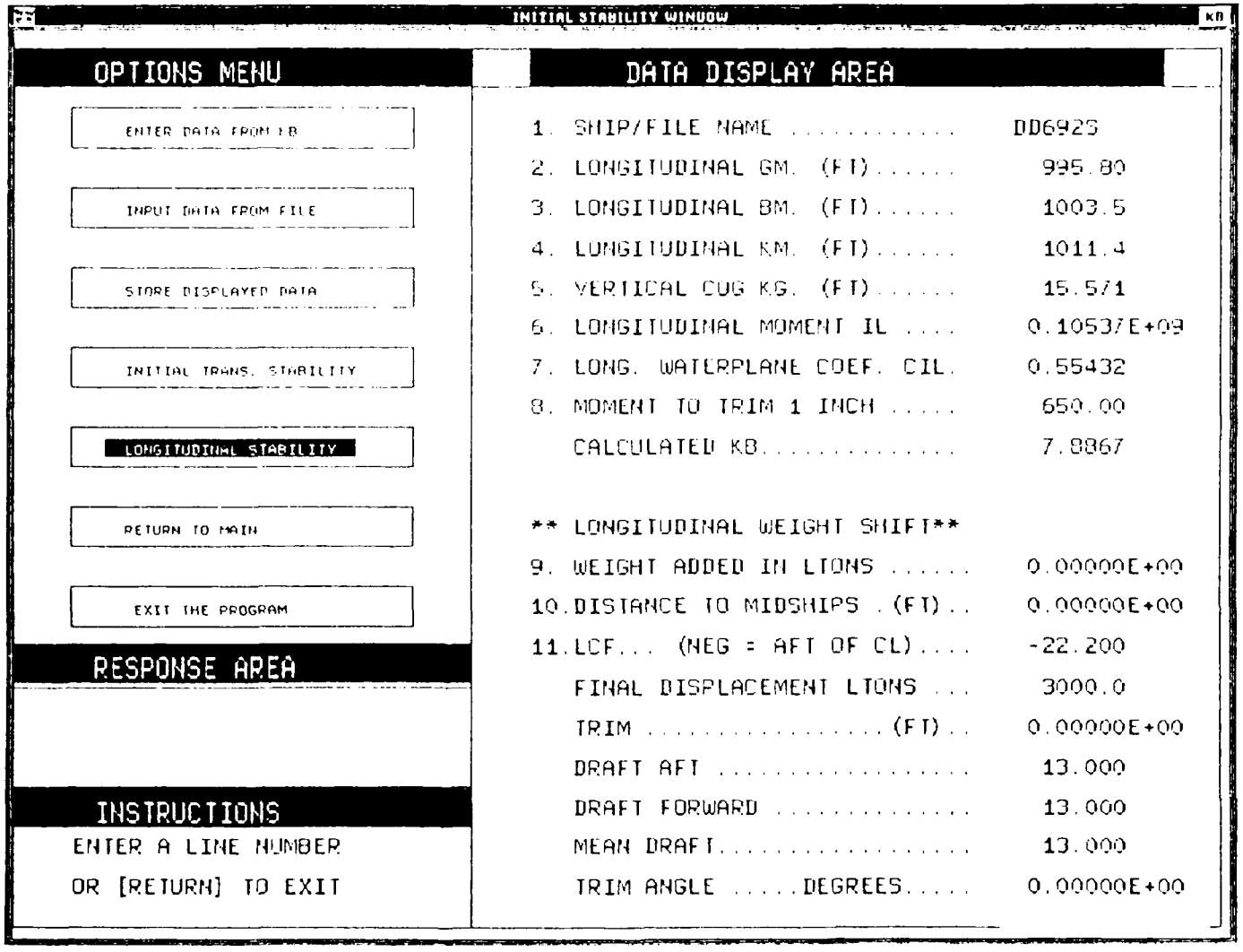

Figure 9 Longitudinal Stability After MTI is Entered

Figure 10 Longitudinal Stability After Weight Shift is Entered

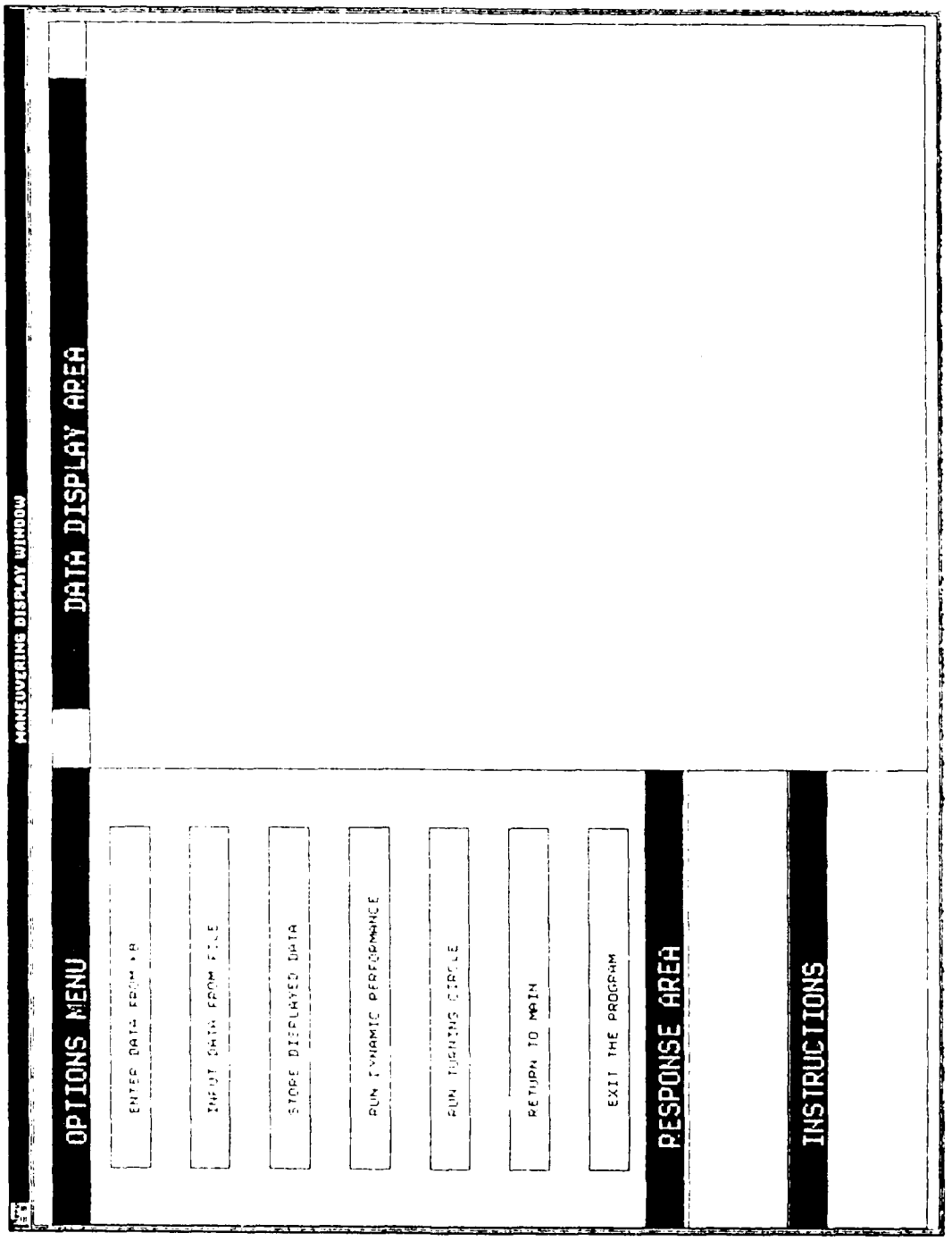

Figure 11 Maneuvering Display Window After Option Selection

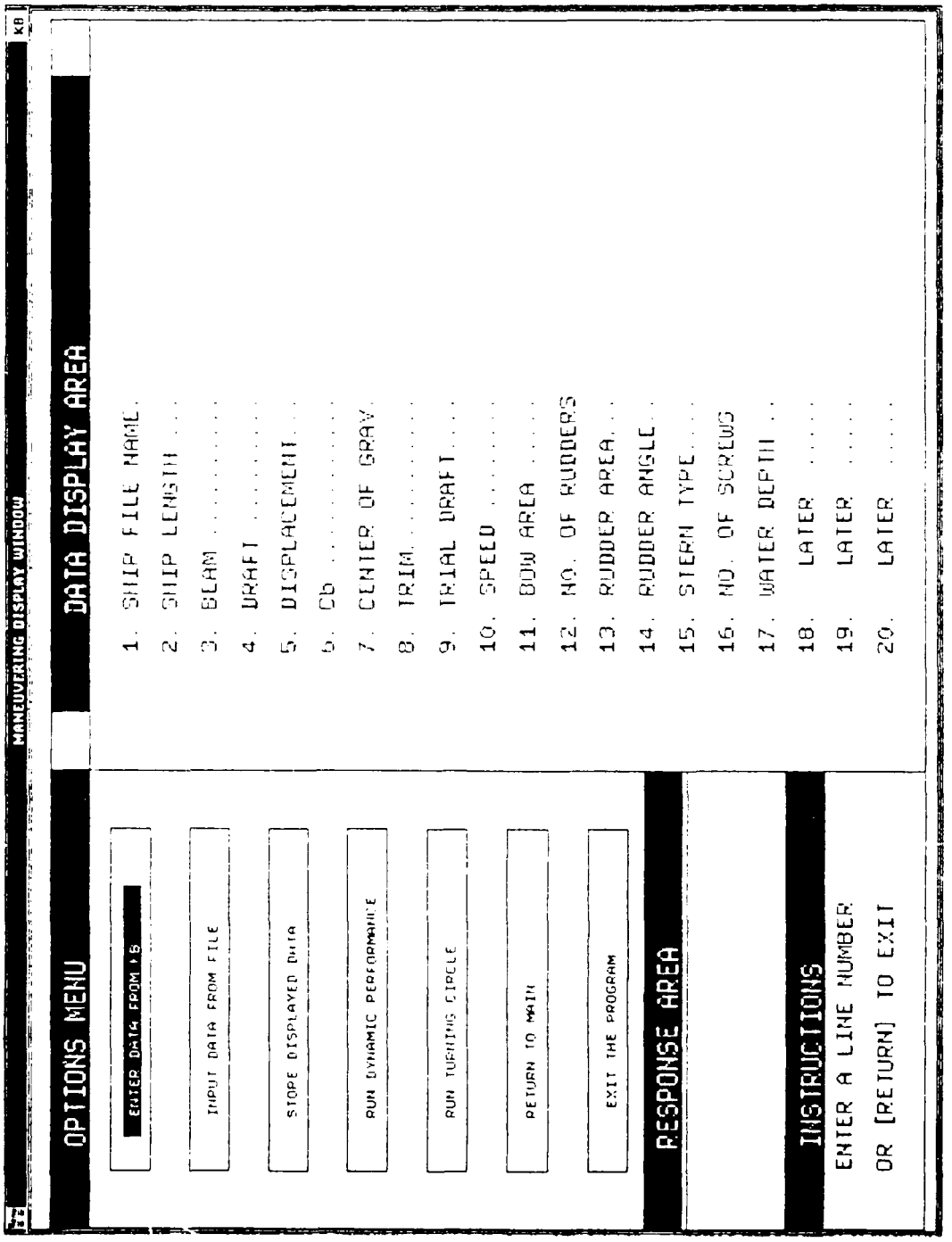

Figure 12 Maneuvering Display ENTER DATA FROM KB Option

Enter the line number one and then enter the Ship File name DD692M. Enter the rest of the data as shown in Figure 13. When all of the data has been entered, exit the option by depressing the [RETURN] key.Next, select the RUN DYNAMIC PERFORMANCE option with the mouse. After a short wait, the display will appear as Figure 14. The program is interactive and any of the first ten parameters can be changed. Enter line ten and change the value of N'd to -.0013. Notice the first caption changed to USER SUPPLIED HYDRODYNAMIC COEFFICIENTS and that the Dynamic performance indicators reflect the change.

Exit the option by depressing the [RETURN] key.

a. Turning Circle

From the Maneuvering Menu, select the RUN TURNING CIRCLE option. After a short wait, the display should change to that of Figure 15. The first eight items can be changed and the effects will be immediately calculated and displayed. At this point, change the rudder angle to 20 degrees and notice the change in TURNING CIRCLE PERFORMANCE. This change does modify the baseline data entered above and will be reflected if the option ENTER DATA FROM KB is selected.

Exit the RUN TURNING CIRCLE option by depressing the [RETURN] key.

b. Store Displayed Data.

A base line ship data set has been entered and the hydrodynamic coefficients were calculated. In addition, a user value of N'd was entered and used to alter the performance indicators.

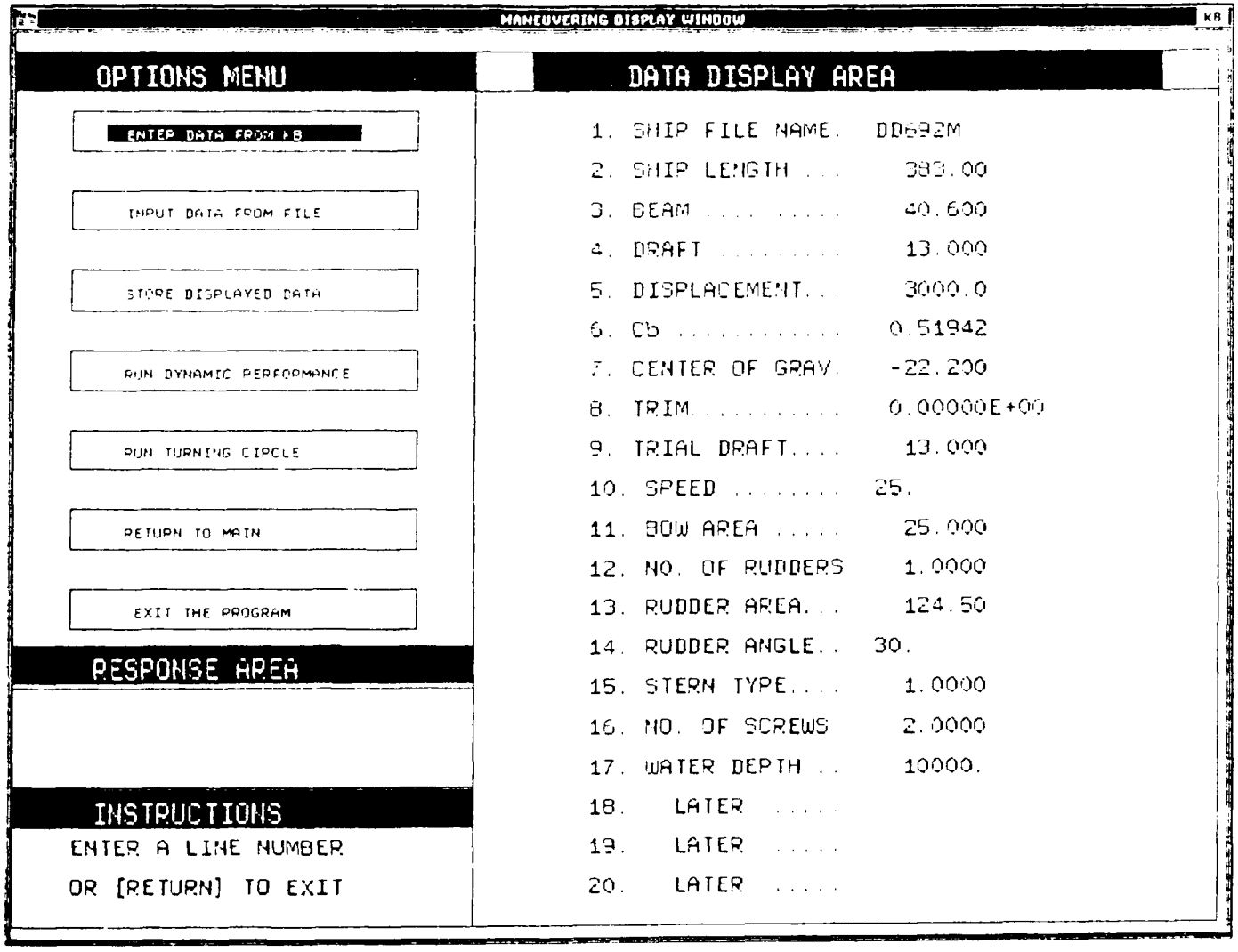

Figure 13 Maneuvering Display Window Showing Example Data

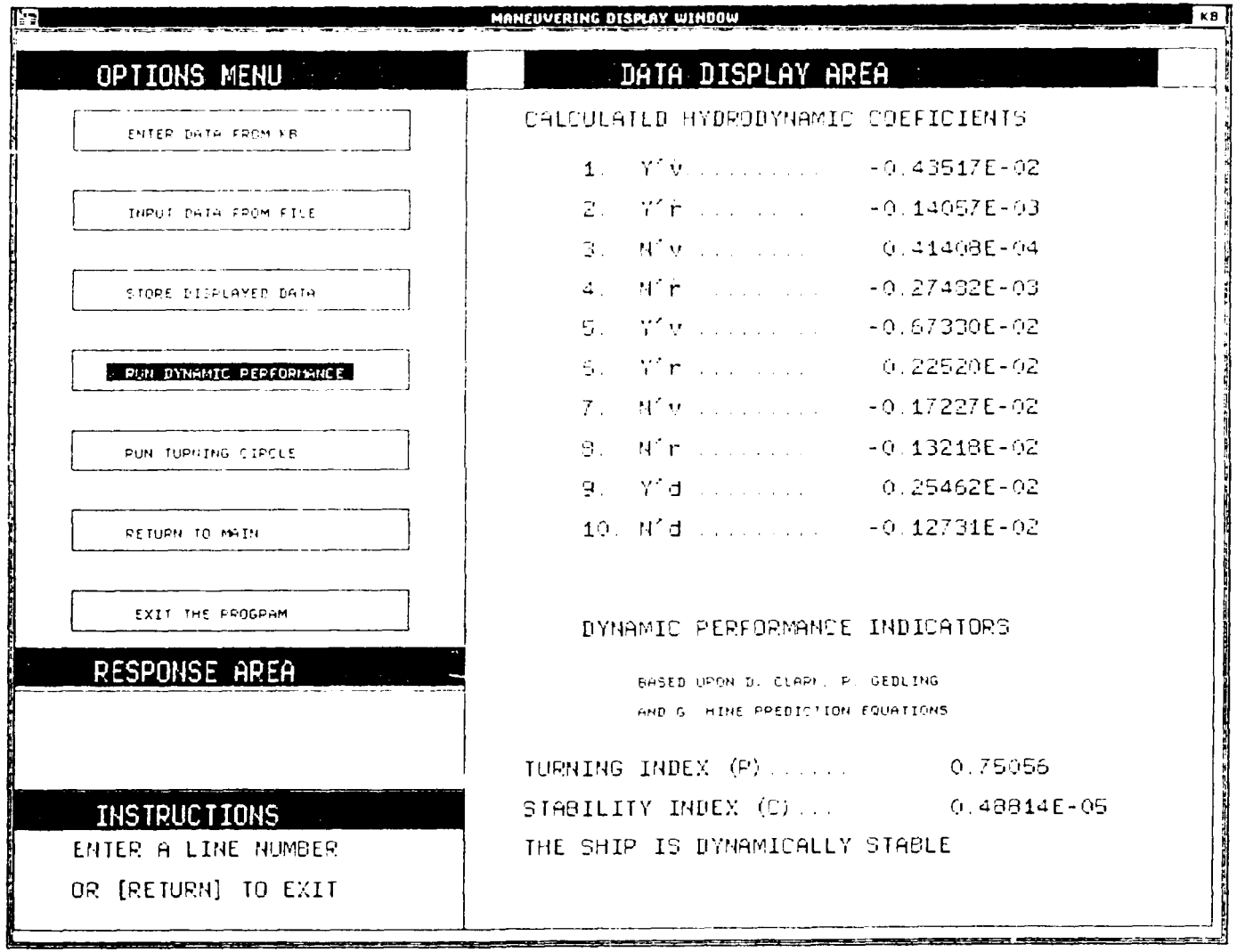

Figure 14 Dynamic Performance Display

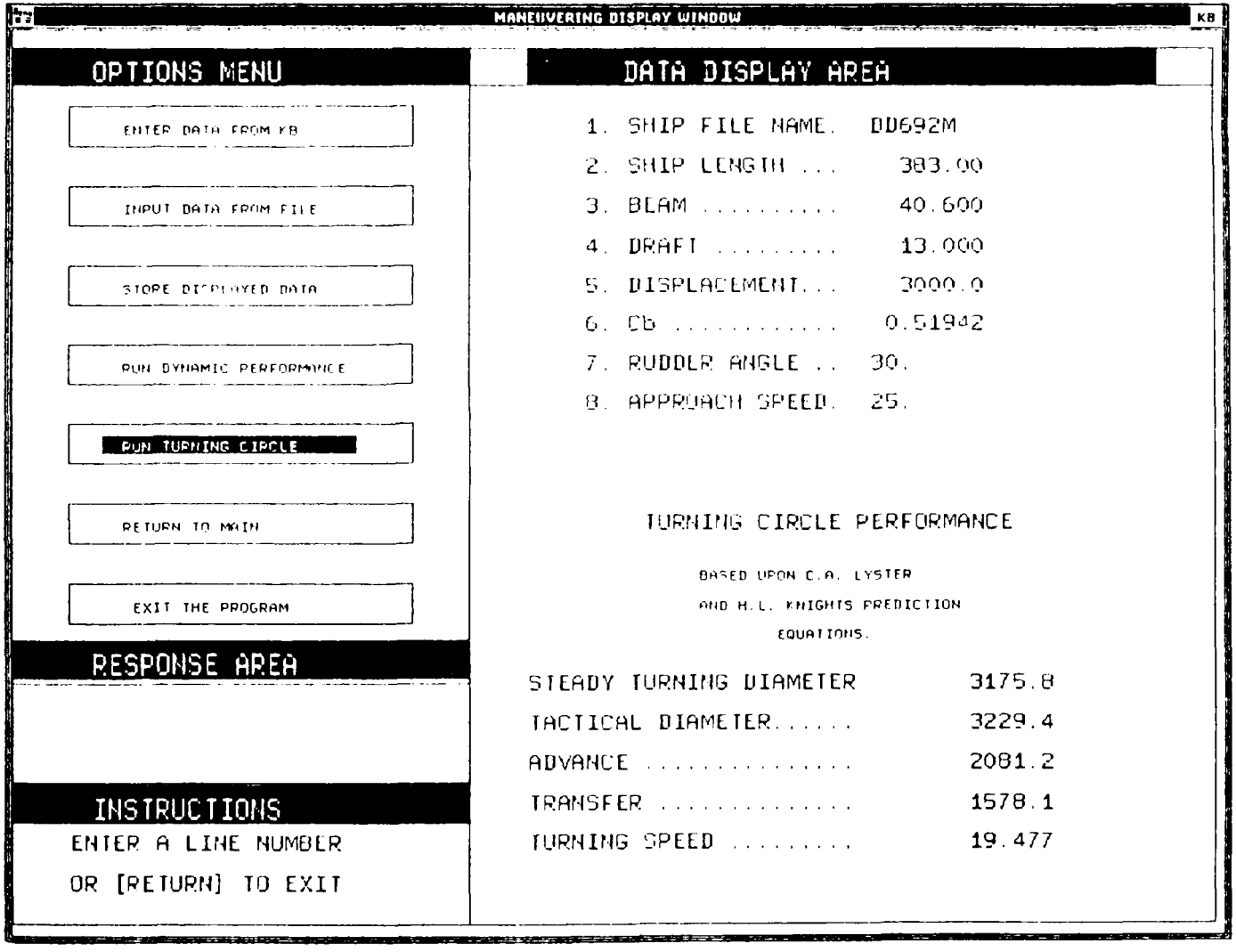

Figure 15 Turning Circle Display

Select the STORE DISPLAYED DATA option with the mouse. The data, including the user specified set of data, is recorded in file DD692M.DAT; and a message to that effect is displayed in the INSTRUCTION area.

At a future time, this file can be recalled into the Maneuvering section using the INPUT DATA FROM FILE option and specifying DD692M.DAT as the file. The stored user supplied hydrodynamic coefficients will not be used in the Dynamic Performance section as the calculated values will be displayed. The user supplied coefficients are only stored for use in the GENERATE REPORT section of the Main Menu.

Exit the MANEUVERING DISPLAY WINDOW by selecting the RETURN TO MAIN option with the mouse.

5. GENERATE REPORT

The files DD692S.DAT AND DD692M.DAT are now in the directory and can be used as input for a written report.

Select the GENERATE REPORT option of the Main Menu with the mouse. The screen will change to that of Figure 16.

a. Creating the Maneuvering Report

Select the CREATE MANEUVERING REPORT option with the mouse. At the prompt, enter the file name and file type DD692M.DAT. This is the name of the data file to use for report creation.

Next, the user is prompted for a file name and file type of the report. At the prompt, enter DD692M.RPT. After a short wait, the option rectangle

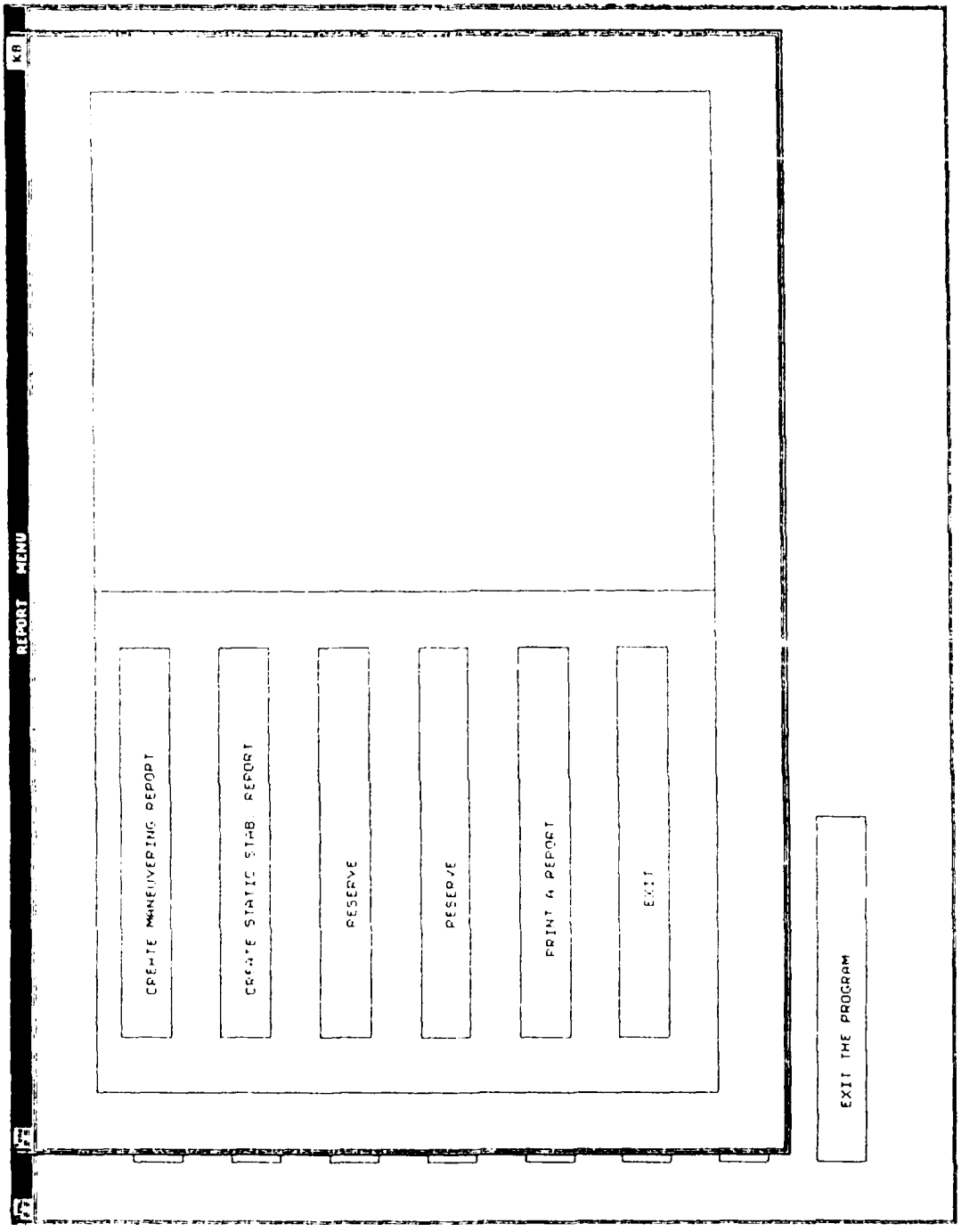

Figure 16 REPORT Display After Option Selection

will return to normal unhighlighted display signifying that the report file was created and can be printed.

Select the PRINT A REPORT option with the mouse. At the prompt, enter the file name and extension of the Maneuvering report, DD692M.RPT. The user is then prompted to enter the name of the printer to be used. This name is the desired printer designation of LASER, LA210_1, LA210_2. LA75_1 or LA75_2. All of these printers produce a 59 line per page printout and work for the formatted report. Enter LASER.

The report and report cover sheet will be printed out on the laser printer and control will revert to the Report Menu. A sample of the report is included as Appendix C. Notice that it contains a section of calculations using the calculated hydrodynamic derivatives and a section with the user supplied coefficients. If no user supplied derivatives are given, the last two pages are not produced.

b. Creating the Static Stability Report

Select the CREATE STATIC STAB. REPORT option with the mouse. At the prompt, enter the data file name and file type to be used to create the report, DD692S.DAT. See Figure 17.

The user is then prompted to enter the file name and extension to be assigned to the report, DD692S.RPT.

The next prompt asks for the number of iterations of TSI to use for calculating Static Stability. The user is then asked for the TSI starting value and

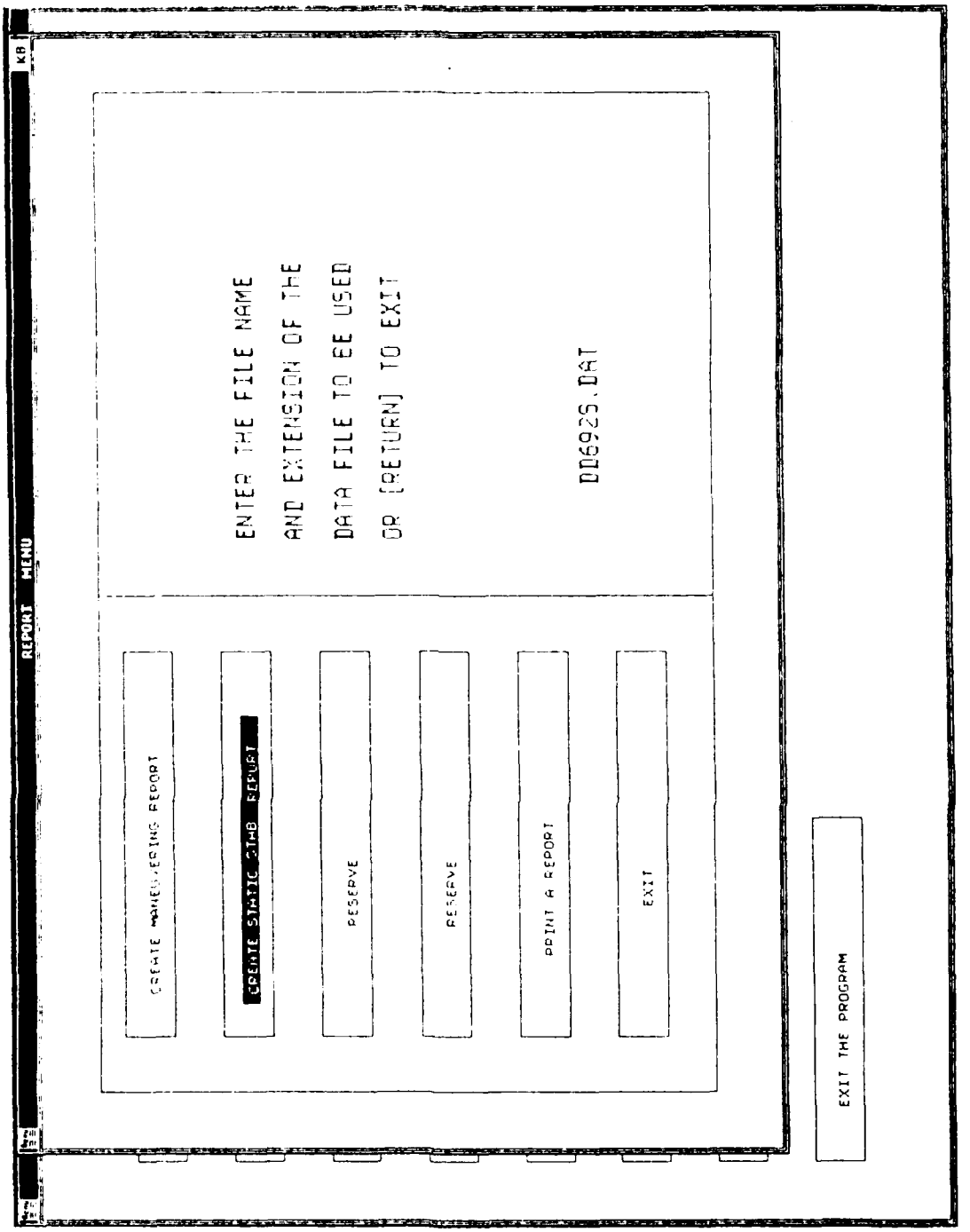

Figure 17 CREATE STATIC STAB. REPORT Display

ending values. For this example, enter three iterations starting at .190 and ending at .194.

The report is then generated and the option is unhighlighted indicating that program control has passed to the Report Menu. Print out the report on the LA75_1 using the same procedure as that used for the Maneuvering Report except using DD692S.RPT for the file name and file type. The Static Report is now printed out on the LA75_1 printer. A sample report is included as Appendix B.

As an aside, any file can be printed from the PRINT A REPORT section as long as the file name plus period, plus file extension, is not more that 12 characters long.

Exit the REPORT SECTION by selecting the EXIT option. Control is now passed to the Main Menu.

6. PROGRAM TERIMINATION

Exit Tool Box by selecting the EXIT THE PROGRAM option. The Main Menu is erased and control reverts to the system window.

III. TOOL BOX PROGRAM DESCRIPTION

A. INTRODUCTION

1. Goals of Tool Box

As previously stated, the main goal of the program Tool Box was to apply VAX graphics routines to the solution of some preliminary naval ship design problems. The program was to be interactive using the system mouse and keyboard. The operator would need minimum ability with the computer system or indeed, any programming language.

The actual operation of the program was to be as easy and as transparent as possible. This is considered important to allow the operator to concentrate on the design problem and not the running of the program.

2. Methods Used

It was decided to use a menu driven problem solving method. The Main Menu, Figure 2, of the program outlines eight major task level options. Each of the major options of the Main Menu leads to a section menu of more task specific options such as Initial Stability Window, Figure 3.

When the pointer is placed in the active rectangle of a menu option, the option is highlighted. If the option is desired, depressing the first mouse button activates the option. In the section menus, the operator is prompted by an instruction for any action required.

In general, the menus suggest the sequence of action to be taken by the operator. As an example, as shown in Figure 2, the first option is to enter data from the keyboard. If the operator has already created a file in a previous session, the next option may be used to enter the data directly from that file. Further options allow storage of current data and computation.

Any changes to data during the session are reflected in the screen output data presentation in real time. This allows the operator to see immediate results of off baseline investigations. Any interactive menu option in the entire program can be exited by depressing [RETURN] or [ENTER] without any data entry. This returns control of the program to the menu which contains the option.

B. MAIN PROGRAM

1. Main Program Tasks

The Main Program, Appendix E, has several functions and is the pivot of the entire program. The main program creates the virtual display boundaries and sets the background and foreground color, as well as setting the program text types. It then draws the basic graphics and initial text for the menus to the virtual display areas. The main program defines the Main Menu option ASTs and sets up ASTs for highlighting choices.

Finally, the main program creates the viewport WD_MAIN and displays the Main Menu. With all preliminary graphics and text drawn and the Main Menu displayed, the system is placed in hibernate mode to await AST activation with the system mouse.

2. Main Menu Subroutines

After the Main Menu is created and active, there are eight options available for selection. At the present time, there are two reserve modules into which nothing has been programmed. The Submarine option does not have a supporting set of subroutines at present. All of these options respond with "OPTION NOT AVAILABLE" when selected.

The flow diagram for the main program is presented in Figure 18.

a. Subroutine STATIC

Subroutine STATIC is the first option displayed on the Main Menu, Figure 2. It is an AST routine which is activated from the main menu by placing the pointer on the STATIC STABILITY rectangle and depressing the first mouse button.

When activated, the Initial Stability Window viewport is created and is displayed on the terminal, Figure 3, as the active window. Subroutine Static then sets ASTs for highlighting and selecting its menu options. With these tasks complete, control is returned to the hibernating system which waits for any option ASTs to be activated. The Static Stability Flow Diagram is shown in Figure 19.

b. Subroutine MANEUVER

Subroutine Maneuver uses the same area of the virtual display as Subroutine Static since the basic functions of data entry, recording and computation are the same. Only the labels of the calculation options are different between the two.

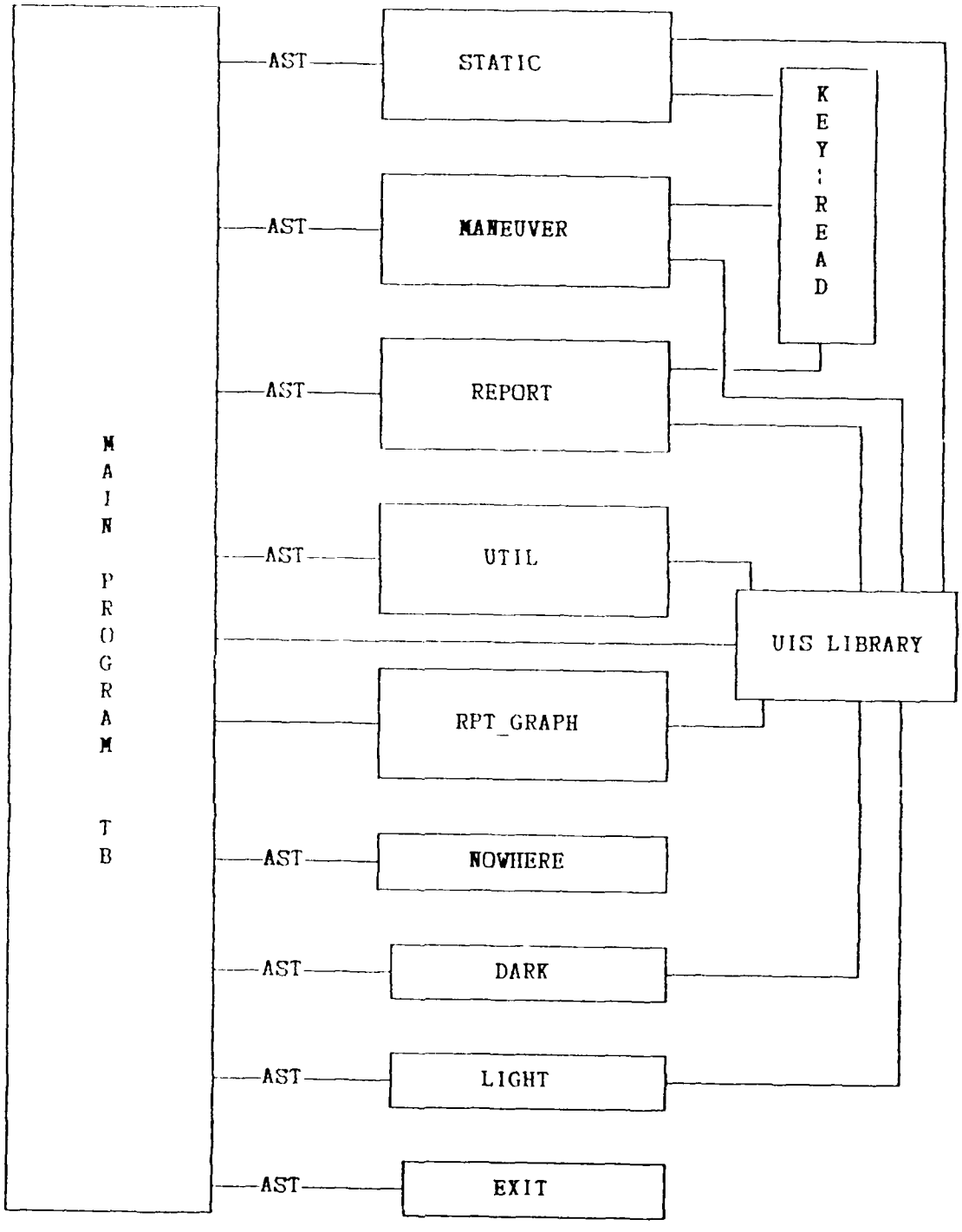

Figure 18 Main Program Flow Diagram

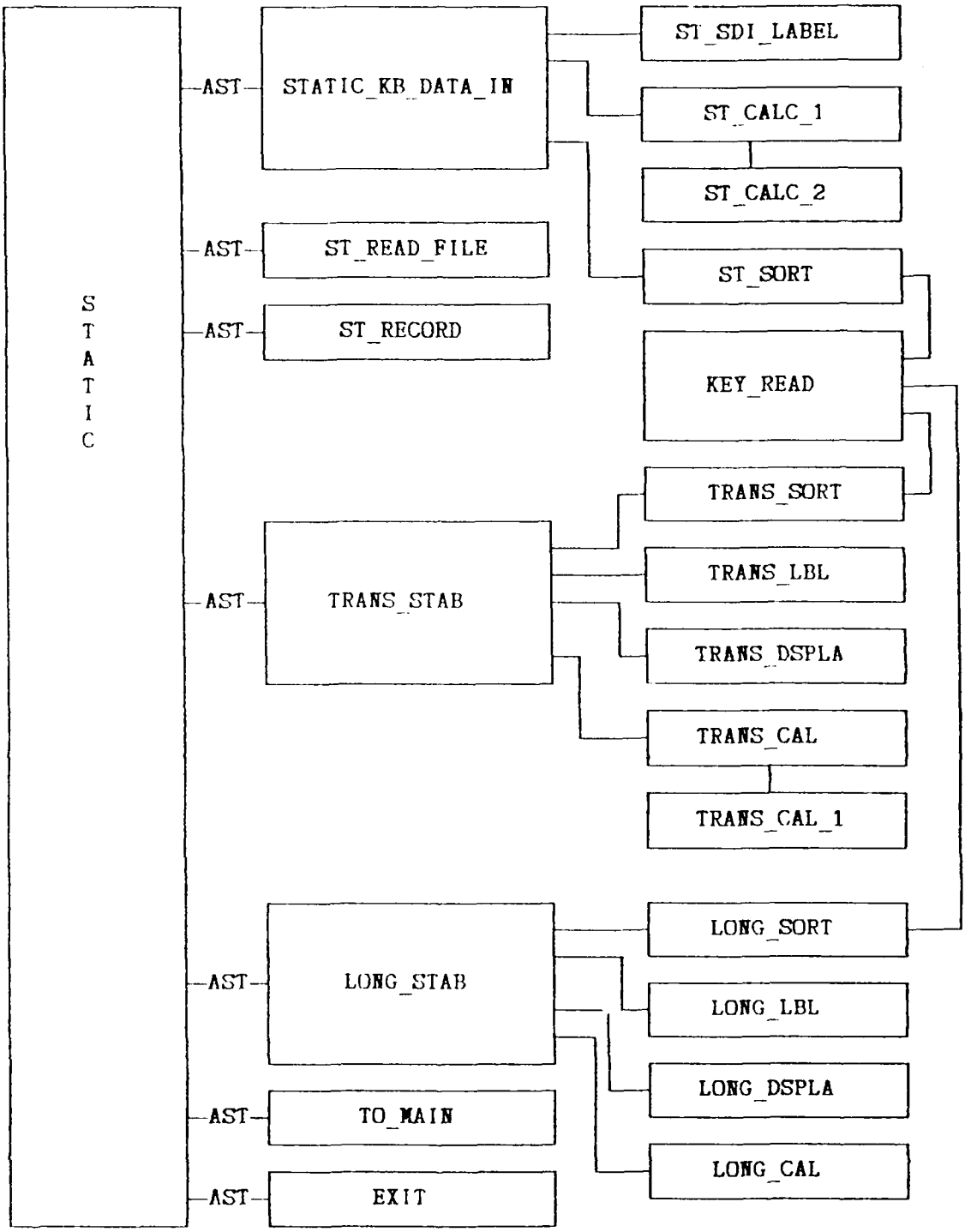

Figure 19 Subroutine STATIC Flow Diagram

Subroutine Maneuver is executed the same as other Main Menu routines and when activated, performs the same functions as the Static Subroutine. The Highlight and Option ASTs are enabled and control is returned to the system in hibernation. The Maneuver Display Window is presented in Figure 11 and the flow diagram is shown in Figure 20.

c. Subroutine REPORT

Subroutine Report provides a detailed hard copy output of either a previously stored Static Stability file or a Maneuvering file. Only the data required to generate the report is stored in the data file.

Subroutine REPORT creates and displays the Report Menu, Figure 16, and enables the keyboard for the all report routines. It then sets Highlight and Option ASTs and returns control to the main program to wait for option selection and activation. Figure 21 is the Report section flow diagram.

d. Subroutine KEY_READ (LINE, FCHAR, XSTSRT,YSTSRT,)*

Subroutine KEY_READ, Appendix F, is the heart of Tool Box in that it is the routine which reads and interprets all keyboard inputs for all of the interactive sections. KEY_READ also knows if the input data is to be character or numerical and does away with the need to enter a period for real data or single quotes for character strings.

A problem was encountered in that the routine KEY_IN = UIS$READ_CHAR(KB_ID) returns a 32 bit longword of which the first byte contains the ACSII key code. The problem was solved by using EQUIVALENCE (KEY_IN,KEYBUF) where KEY_IN was defined as a LOGICAL*4 and

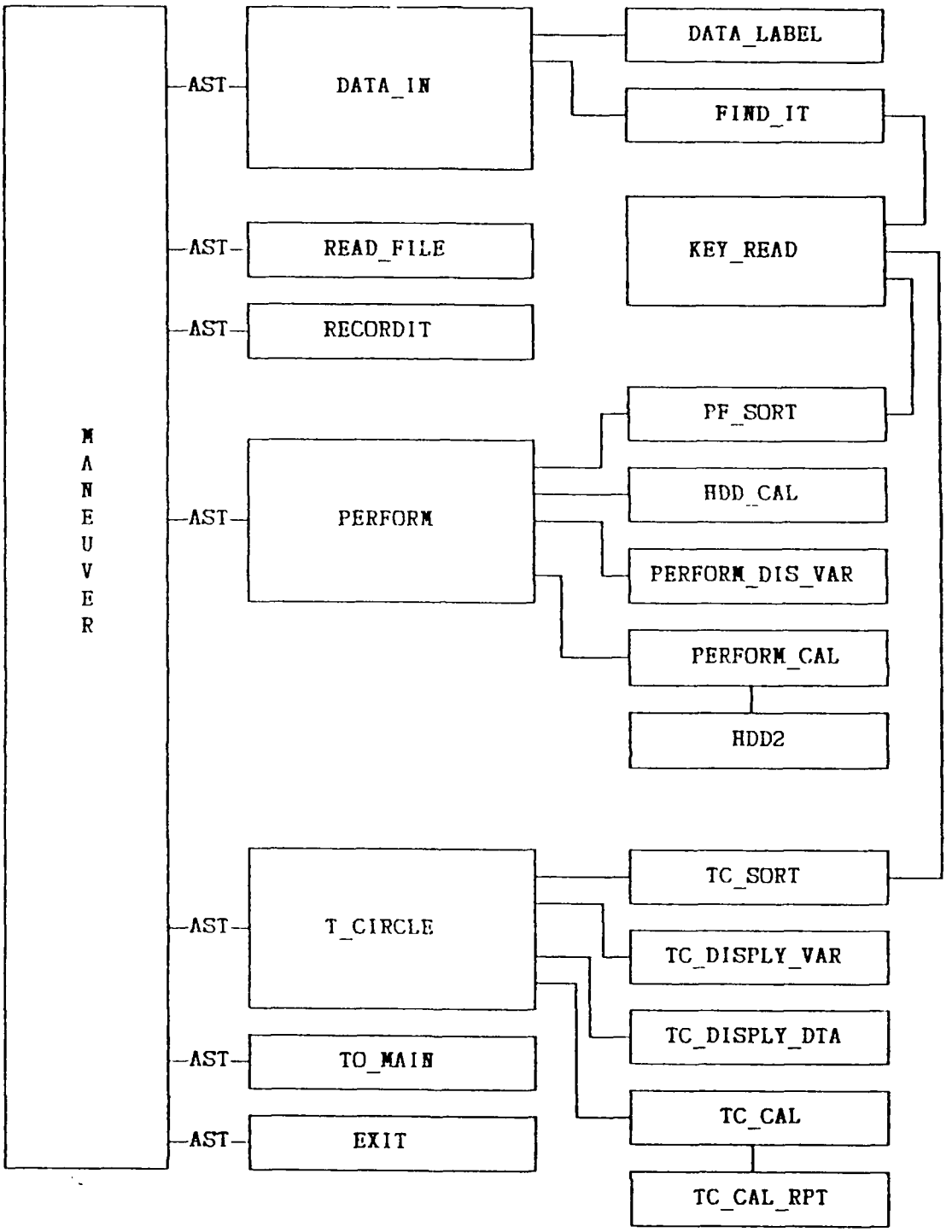

Figure 20 Subroutine MANEUVER Flow Diagram

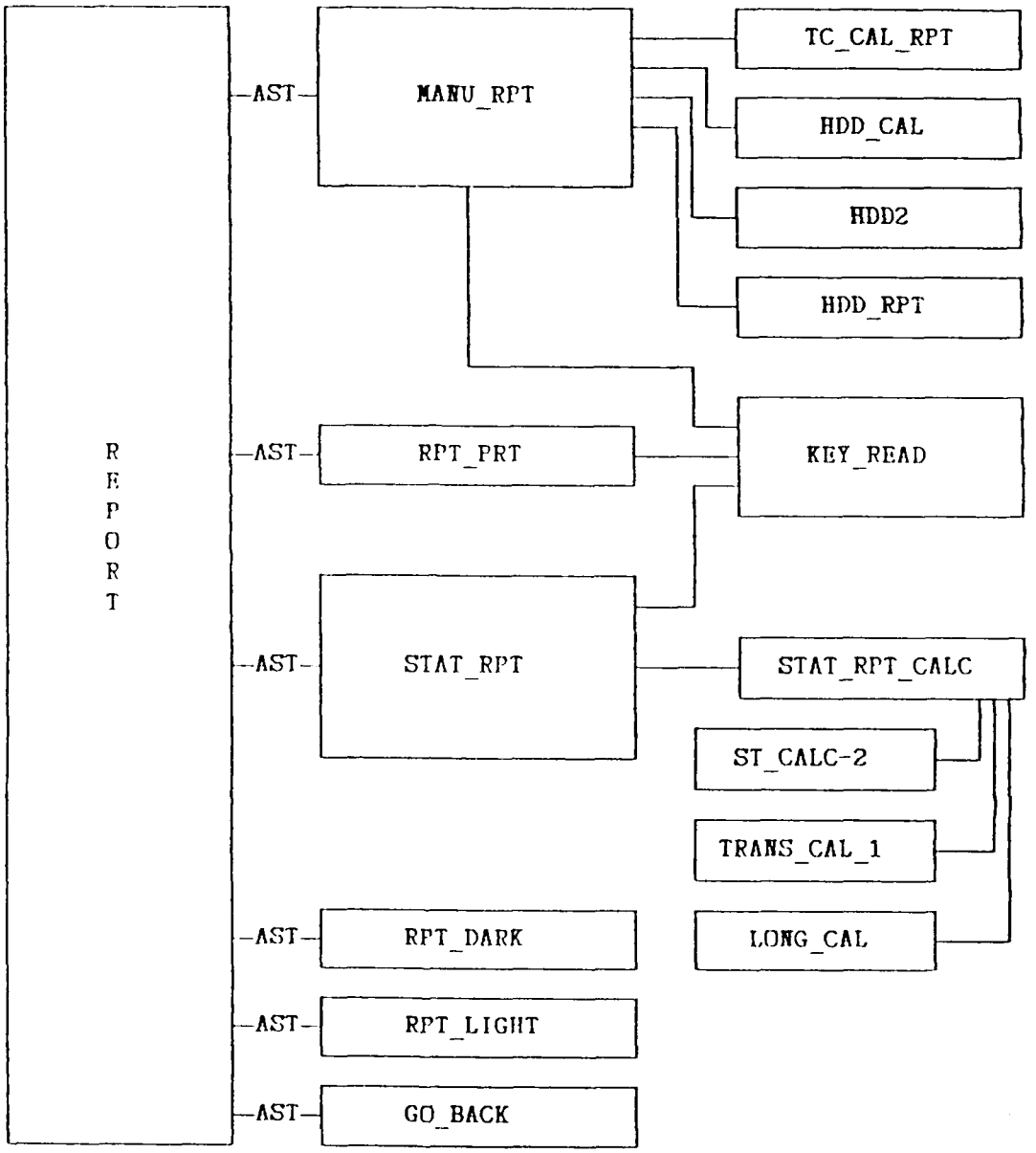

Figure 21 GENERATE REPORT Section Flow Diagram

KEYBUF was defined as LOGICAL*1 KEYBUF(4). In this way, the ASCII keycode is stripped using KEYBUF(1). The keycode is then converted to a character using CALL LIB$CHAR(KEY_IS,KEYBUF(1)).

The character string is built by adding each successive keystroke to a buffer called LINE. LINE is an input character string which is passed from KEY_READ to the calling subroutine. If the string is to be converted to real data, LINE is conditioned with a period in the appropriate position.

FCHAR is the Character string/floating point flag. If FCHAR is true, then the input string is to be converted to floating point. During input, the string is continuously checked for a period. When the return key code is read, if a period has not been supplied to a floating point candidate, one is added after the last character entered. The conditioned string is then returned to the calling routine.

KEY_READ also provides backspace and keystroke echoing to view the input before actually entering the data with the return key. When a backspace keycode is detected, the program deletes the last character from LINE and corrects the echo display. If the return key is depressed without any data entry, the program exits the routine by the alternate return specified by LINE_NUM in the calling subroutine.

If the calling program has requested numerical data, when LINE is returned it is converted to a floating point number using the statement READ (LINE,FMT='(G12.5)') VARIABLE. The G descriptor is used to make the program more flexible for data input. In the cases where integer data is used, the real number is then converted using INT(REAL).

After calculations are performed in the program, the numerical results must be converted to character strings for display on the virtual display, since only text can be written to the graphic screen. This is accomplished using the statement WRITE (CHAR_VARIABLE, FMT = '(G12.5)') REAL_VARIABLE. [Ref. 4]

e. Subroutine UTIL

Subroutine UTIL is a short routine which allows the user to run nearly all DCL commands from within the program, without loosing data or time. Many other applications require leaving the program to perform file maintenance such as printing, renaming, deleting or viewing a directory. The subroutine does not create its own display, but uses the regular workstation display. The user is prompted for DCL commands to enter. The character string is assigned to the variable COMMAND and executed using the LIB$SPAWN(COMMAND). To exit the Utilities option, the [RETURN] is pressed without any other entry. This causes the main Menu to be popped to the front and become the active screen. Subroutine UTIL is presented in Appendix G.

f. Subroutines DARK AND LIGHT

Whenever the pointer moves across a boundary into an option area, the AST DARK is activated. Subroutine DARK reads the pointer position and calculates the option number. It then redraws the option text in reverse video to highlight the choice.

When the pointer exits the rectangle, subroutine LIGHT causes the option text to be redrawn in the original text, de_emphasizing it. Other subroutines

with similar names, (LIGHT2, DARK2, RPT_DARK, RPT_LIGHT) are copies of DARK and LIGHT and perform similar functions for the other parts of the program. Copies are used with different variable names in order to avoid numerical confusion within the program if two ASTs are active at one time on the system stack. In this way, the correct option is de_emphasized upon exiting the current option.

g. Subroutine NOWHERE

This routine simply informs the operator that the option chosen is not yet available and returns control to the Main Menu.

h. Subroutine EXIT

The EXIT subroutine expands the workstation window to its regular working size, deletes the virtual display, calls SYS$WAKE() and returns control to the Main program. The first instruction after SYS$HIBER() is the END statement which terminates the program.

C. DESCRIPTION OF FUNCTIONAL GROUP SUBROUTINES.

Since many of the basic functions performed by the subsections were very similar, an attempt was made to build basic functional routines and then modify them to perform the more specific function required by a subsection. In this way, a total of seven functional routines were written and modified to cover 25 specific functions.

In the following discussion, only one of the functional subroutines will be discussed in detail and the similar routines will be identified. Finally, the individual subroutine differences will be discussed. A copy of the subroutine under discussion will be found in the appendix for reference.

1. Subsection Initializing and Control ASTs

The first functional group is the Subsection Initializing and Control AST Group. These subroutines are the ASTs which are enabled whenever the program enters the Static Stability Menu or the Maneuvering Menu. This group consists of

- STATIC_KB_DATA_IN
- TRANS_STAB
- LONG_STAB
- DATA_IN
- PERFORM
- T_CIRCLE

Subroutine DATA_IN is the data entry initializing and control AST for the Maneuvering Subsection and was the prototype subroutine. All others in this group evolved from it. Subroutine DATA_IN is presented in APPENDIX H.

As in all of the graphics subroutines, the first lines call in the UIS libraries and set up the section type and common statements. The program then moves the pointer out of the active AST region of the display to prevent double AST activation. Next, the virtual keyboard is created and enabled for the window being displayed.

At line one, the instruction area is erased and the first instruction, "ENTER A LINE NUMBER OR [RETURN] TO EXIT", is written to the screen. At line three, the subroutine KEY_READ is called to read in the line number of the data to be entered or changed.

The arguments of the call to KEY_READ are the character string LINE and the starting position, world coordinate pair (0.,12.), for key echo. This call also specifies an alternate return line number of 300.

If the [RETURN] key is depressed without any data entry, the subroutine is cancelled and control returns to the main program. The logical .TRUE. informs the subroutine that the character string LINE needs conditioning for conversion to real numbers.

Following line three, the string LINE is read into the real number RLINE_NO. It is then converted to an integer LINE_NO.

LINE_NO is used as a branching argument in the computed GO TO statement. If the LINE_NO is out of range, "IMPROPER LINE NUMBER, ENTER A NEW LINE NUMBER PLEASE" is displayed in the instruction area. Control then returns to line one.

Each of the next series of line numbers is ten times the corresponding line number of the displayed data. Each is a functional section by itself.

First, the line number subsection calls a sorting and controlling routine which sets conditions for the data input, displays the proper instruction and reads back the string LINE which contains the requested keyboard data. This call also specifies line 300 as an alternate return which allows exiting the subroutine in the same way as described above.

The text string LINE is stored in the appropriate character variable and displayed on screen. If required, it is also converted to real data using the formatted read statement. If the input data alters other data which is currently

displayed on the screen, the values are recalculated, converted to character strings and displayed. This can be seen in groups at lines 20 through 50, which will recompute either displacement or block coefficient if any of the parameters are changed.

After each line number section is completed, the subroutine returns to line one and requests a new line number of data to be changed or entered. If the [RETURN] is depressed without a line number, the program goes to line 300.

The exiting sequence begins at line 300 where the keyboard is disabled. Next, the instruction area is erased and the new instruction to select an option with the mouse is written to the screen. The routine then returns to the main program in hibernation.

2. Sorting and Instruction Routines

The next group of subroutines are called from the Initializing and Control ASTs. The Sorting and Instruction routines find the appropriate instruction and display it in the instruction area. They also prompt the KEY_READ subroutine for data input and display the returned data in the appropriate place on the display screen.

The Sorting and Instruction functional group is made up of the following subroutines:

- ST_SORT
- TRANS_SORT
- LONG_SORT
- PF_SORT

- TC_SORT
- FIND_IT

Subroutine FIND_IT is presented in Appendix I. All of the other Sort subroutines perform the same functions in very similar manners.

Subroutine FIND_IT first includes the file TOP.FOR with the general libraries and common statements. It then defines several more data types for use in the subroutine.

Using the input LINE_NO, the Y coordinate, DEL_Y, of the data to be displayed is calculated. Next, the instructions for the particular data requested are displayed.

An IF block then decides, based upon LINE_NO, if the data requested is to be converted to floating point data. If it needs conversion, the FCHAR flag must be set to .TRUE. for use in the KEY_READ subroutine.

Subroutine KEY_READ is then called for the input data which is transferred as the string LINE. FCHAR, the conditioning flag, and the world coordinates of the key echo location are also sent to KEY_READ. As discussed above, an alternate return is provided for exiting the routine if no data is entered before [RETURN] is depressed.

When subroutine KEY_READ returns control, the old data is erased and the new data is drawn to the display screen. Control of the program then returns to the calling AST which asks for a new line number.

3. Labeling Routines

There are six different screens of information presented in the Static and Maneuvering sections of the program which require six different sets of display characters. The data display area has a total of twenty usable lines to display data. Some of the lines will contain changeable data, which are characterized by an associated line number. Any other information displayed is informational or program generated data.

The labeling subroutines are called once at the beginning of each of the section ASTs to draw the unchanging portions of the particular display screen. The subroutines are:

- LONG_LBL
- TRANS_LBL
- ST_SDI_LABEL
- TC_DISPLY_VAR
- PERFORM_DIS_VAR
- DATA_LABEL.

Subroutine DATA_LABEL will be discussed in detail and is presented in Appendix J.

Subroutine DATA_LABEL is called from AST DATA_IN and is fed arguments containing the virtual display identifier and the character array containing existing data. The subroutine declares all variables as implicit integers, loads the UIS libraries and declares subroutine specific character and real types.

The first step in the creation of the display screen is to erase any previous information drawn to the Data Display Area. The twenty lines of the new display are then given line labels. The line labels are then drawn to the virtual display and the screen using a twenty step Do loop which also writes up any previously entered data strings. After the text writing loop, the subroutine exits back to the calling AST.

Two of the Labeling subroutines, TC_DISPLY_VAR and PERFORM_DIS_VAR do not write labels for the entire display area. These two routines only write the labels for the variable data portion of the routine. The remaining display area is labeled and drawn as part of the associated calculating subroutine.

Although this gives functional division to the program, it does waste some computer time in that the entire result section of the screen is rewritten each time a variable is changed. It is felt that the method used for the DATA_IN is the better programming technique.

There are two more routines which fit into the data display category and they are TRANS_DSPLY and LONG_DSPLY. These routines are used to display the results of changing an input variable on other changeable input data in the transverse and longitudinal stability sections.

The subroutines are called after the calculations are performed and the results are written into character variables, then drawn to the display at the proper location. The subroutines then exit.

In other sections, this function is performed either in the calculating subroutines or in the calling AST itself. This separate subroutine method is more functionally readable, but can waste space if only a few variables are changed by data entry.

4. Calculating and Conversion Routines

The actual calculations performed in the sections are performed by the Calculation and Conversion routines. These routines usually are in pairs. One routine (a conversion routine) calls a second which performs the actual computations and returns the results. The converting routine then writes the floating point information into character variables for screen display.

The calculating routines in Tool Box are ST_CALC_1, ST_CALC_2,TRANS_CAL, TRANS_CAL_1, LONG_CAL, PERFORM_CAL, HDD_2, HDD_CALC_CAL AND TC_CAL_RPT. The calculating pair TC_CAL and TC_CAL_RPT are presented in Appendix K.

The UIS libraries are read into the subroutine and the type statements are declared. A five element, 12 character, string array TC_DATA(5)*12 is declared for holding the text representations of calculated data. The Subroutine TC_CAL_RPT is called to perform the actual calculations using data previously entered.

The returned data is converted to character strings using the command; WRITE(TC_DATA(1), FMT = '(G12.5)') FLOATING_POINT .[REF. 4:p. 6-21]

After all data has been converted, it is drawn to the virtual display and the screen. The subroutine then exits back to the calling AST.

The actual mathematics involved in the computing portion of the subroutines are well understood and are not presented. All of the static stability relationships were derived from [Ref. 4] and Principles of Naval Architecture [Ref. 5].

Derivation of the basic equations for maneuvering stability from Clarke, Gedling and Hine, [Ref. 6], and the Turning Circle Performance derived from Lyster and Knights [Ref. 7], are presented in Appendix A.

5. Data File Recording Routines

Whenever data presented on the screen is to be saved, one of two ASTs is activated. AST RECORDIT records the necessary data for the Maneuvering Performance section and AST ST_RECORD performs the function for the Static Stability section. The information available at the time of the AST activation is recorded in a formatted file using SHIP NAME as the file name and DAT as the file extension.

Subroutine RECORDIT is presented as Appendix L and is discussed in detail below.

The files TOP.FOR and TOP_MANU.FOR are read into the file and subroutine type statements are declared. Next, the pointer is moved to an inactive part of the screen and the keyboard is activated.

The file is opened with a Status of 'NEW' and the data is read in formatted form into the file. An ENDFILE command is given as the last input to the file and it is then closed.

Successful completion of the event is reported by concatenating the FNAME with DAT and writing that the file was saved as FNAME.DAT to the screen. Control is then passed to the main program.

6. Data File Reading Routines.

There are two AST subroutines written for reading previously stored data into a program section. These are READ_FILE and ST_READ_FILE. Subroutine READ_FILE is presented in Appendix M and discussed below.

If a file has been previously created, its contents can be recalled for modification using the READ_FILE routine. In the subroutine, after the files TOP.FOR and TOP_MANU.FOR are included, the pointer is repositioned to an inactive portion of the screen. The keyboard is then activated and the instruction to, 'ENTER THE FILE NAME AND FILE EXTENSION' is written to the screen.

Subroutine KEY_READ is called to read in the file name keystrokes in the form of the character string, LINE. Next, the file is opened; and the formatted data is read in and assigned to appropriate variables.

After the last data read, the file is rewound and the data is read in again, but this time into a character array. As previously noted, this is required because only text can be drawn to the display.

The file is then closed and the keyboard is disabled. The instruction area is then erased and the new instruction to select an option with the mouse is written to the screen. Before exiting, Subroutine DATA_LBL is called which writes the newly created character data to the screen. The AST then returns control to the main program.

IV. VAX VMS GRAPHICS ROUTINES

A. GENERAL

The vax graphics routines are a group of routines which are called from a high level programming language. The routines are located in libraries which must be included in the program before any of the routines can be accessed. The following calls must be in a FORTRAN program or subroutine before either one can call UIS routines.

- INCLUDE 'SYS$LIBRARY:UISENTRY'
- INCLUDE 'SYS$LIBRARY:UIS$USRDEF'

The INCLUDE statement is a very powerful command which will be discussed later.

B. GRAPHICS COORDINATE SYSTEMS

The VAX VMS uses graphics routines which define two dimensional graphics displays using two different sets of cartesian coordinate systems, world and absolute. [Ref. 8]

The world coordinate system is a user defined system which serves as the reference for virtual displays. It is independent of device and unit of measurement. This allows a choice of coordinates which are convenient to the problem and the design of the display. The world coordinates are used as references for graphics and text applications within the virtual display.

The boundaries of the graphics are specified with the creation of the virtual display. The range should be large enough to accommodate the expected problem plus an added amount for unexpected growth. The allowance for growth is particularly needed if the graphic will contain text. Changing the size of the virtual display after initially aligning and sizing the text within it, will usually distort the earlier text lines, requiring readjustment.

Output devices have defined physical dimensions and therefore need a set of coordinates specific to each device. These coordinates are called absolute device coordinates. In particular, they specify positions on the MicroVMS workstation display in centimeters with respect to the lower left hand corner of the screen.

Several graphics routines allow specification of exact placement on the screen with absolute coordinates. Omission of coordinates in those routines which allow specification of the absolute coordinates causes the system to use default placement. All of the Tool Box routines use the defaulting system values.

All coordinates are given as X Y pairs with X as the traditional horizontal reference and Y the vertical. The X Y pairs are always given as floating point numbers. Not specifying a floating point coordinate usually results in the value defaulting to zero.

C. CREATING A GRAPHICS DISPLAY

1. General

A graphics display is started by first creating a virtual display. This display is a conceptual display which has a set of confining coordinates but no

physical dimensions. It could be thought of as a large piece of blank paper upon which graphics and text are to be drawn. [Ref. 8:p 8-1]

After the virtual display is defined, graphics routines draw and write to the display. To view the display, windows are created to various parts or all of the virtual display. Windows can be opened, closed, resized, moved and deleted without affecting the virtual display.

The window is associated with a display viewport, which is the area of the physical display screen allocated to the window. There can be any number of viewports displayed on the screen at any time. Viewports overlapping each other cause occlusion, with the last viewport created being on top. The relationships between virtual display, window and screen display are illustrated in Figure 22.

2. Display Routines

a. VD_ID = UIS$CREATE_DISPLAY (X1,Y1,X2,Y2,WIDTH,HEIGHT,VCM_ID)

This routine defines the boundaries of the virtual display and its actual viewport size on the workstation screen [Ref. 8:p 18-26]. It will be one of the first statements in the graphics program. The default colormap of two entries defines the background (white) and foreground (black) colors. The colors can be changed using UIS$SET_COLORS() [Ref. 8:16-1].

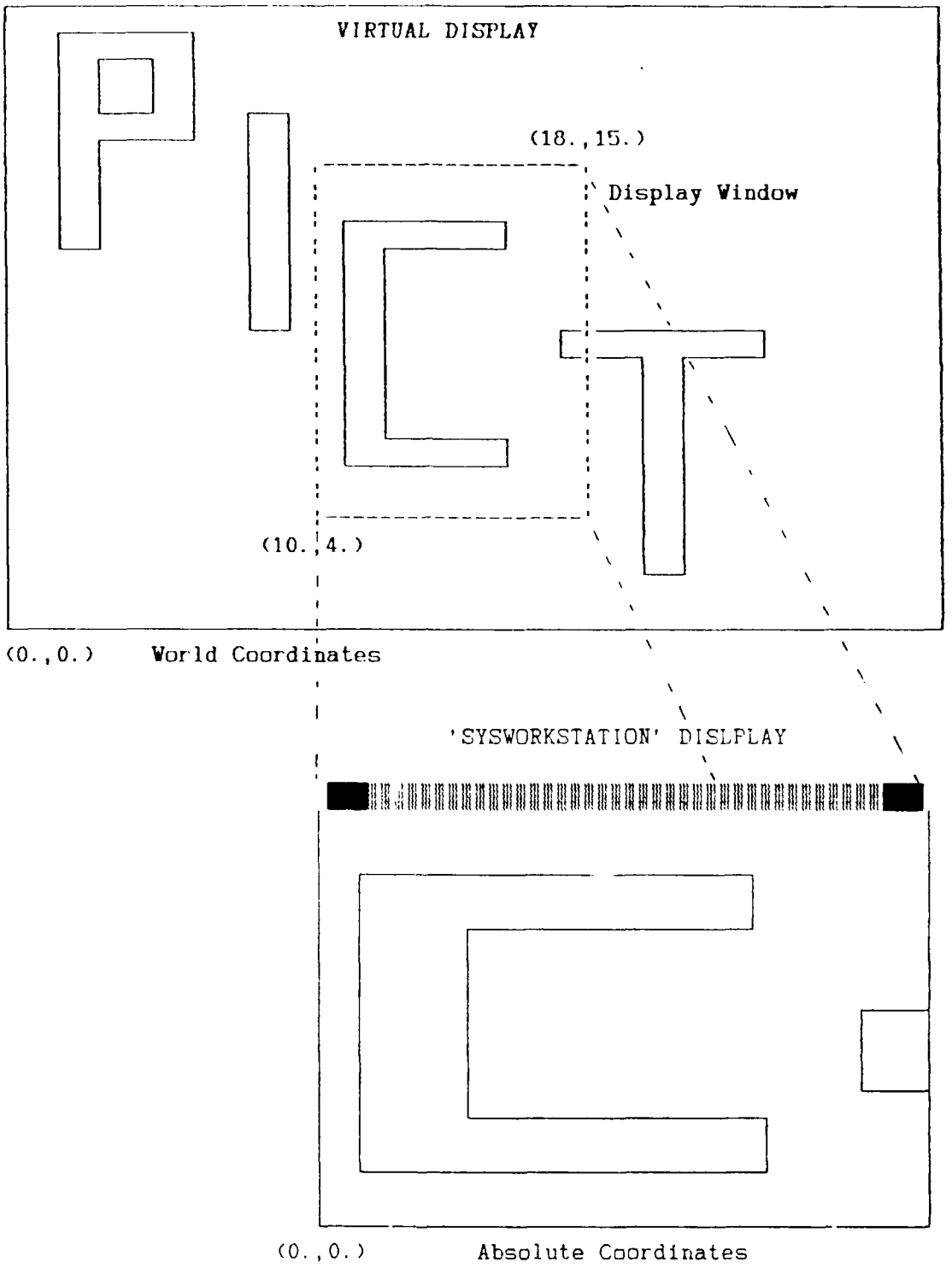

Figure 22 World Coordinate to Absolute Relationship

- VD_ID is a longword value, uniquely identifying the display.
- X1,Y1 are the lower left world coordinate pair.
- X2,Y2 are the upper right world coordinate pair.
- WIDTH is the display viewport width in centimeters.
- HEIGHT is display viewport height in centimeters.
- VCM_ID is the virtual colormap identifier. This need not be specified. If not specified, a default colormap of two entries is created.

b. WD_ID = UIS$CREATE (VD_ID,'SYS$WORKSTATION',TITLE, X1,Y1,X2,Y2,WIDTH,HEIGHT,ATTRIBUTES)

The display window routine creates a viewing window of the virtual display. World coordinates are used to define the scope of the viewable area of the virtual display. If the coordinates are not used, the window defaults to view the entire virtual display. [Ref. 8:p 18-37]

- WD_ID is a returned longword value which uniquely identifies the window. It is the name of the window.
- VD_ID is the name of the virtual display.
- 'SYSWORKSTATION' is the name of the output device. This indicates that the output device for this graphics window is the terminal screen.
- TITLE is a character string title. It will appear in the window banner area. It need not be specified.
- X1,Y1 is the world coordinate pair of the lower left corner of the desired display area. It need not be specified.
- X2,Y2 is the world coordinate pair of the upper right corner of the desired display area. It need not be specified.

- WIDTH, HEIGHT are the dimensions in centimeters of the display on the terminal. If not specified, the height and width used in the virtual display are used.
- ATTRIBUTES were not used in the programming and are not required.

When created, the system will attempt to display the viewport in an unused portion of the screen. If there are multiple viewports and interference occurs, the last created window will overlap (occlude) the earlier windows.

c. UIS$DELETE_DISPLAY (VD_ID)

The delete display routine deletes the display named VD_ID and disables all calls using the VD_ID name. [Ref. 8:p 18-48]

d. UIS$DELETE_WINDOW (WD_ID)

The delete window display deletes the viewport from the screen and disables all calls using the WD_ID name. [Ref. 8:p 18-54]

e. UIS$PUSH_VIEWPORT (WD_ID)

The push viewport routine causes the named viewport to be at the back of the viewing screen. Any other displays present will occlude the pushed display if there is overlap. This is a useful routine to switch the active menu to the back in order to make another menu active at the front of the screen. [Ref. 8:p 18-239]

f. UIS$POP_VIEWPORT (WD_ID)

The pop viewport routine does nearly the reverse of the push routine. It brings the named viewport to the front of the screen to become the

active screen. If there is overlap, this popped screen will occlude any other display present on the screen. [Ref. 8:p 18-234]

g. UIS$SET_COLOR (VD_ID,INDEX,R,G,B)

The set color routine sets the color of the listed index to combinations of red, blue, and green for graphics and text presentation. This program did not specify a colormap in the virtual display creation; and so there are only two indexes available: 0 (background) and 1 (foreground).

The desired colors are displayed as a mix of red (R), green (G), and blue (B). Each color can have a value between zero and one (0 to 100% of the color present). As an example, a mix of R = 0., G = 0. and B = 0. would give black, while R = 1., G = 1. and B = 1. gives white. A mix of R = .75, G = 1. and B = 1. gives a light bluish green color. [Ref. 8:p 18-283]

D. CREATING GRAPHICS ON THE VIRTUAL DISPLAY

There are seven UIS routines which draw graphics on the virtual display. Of these seven, only UIS$LINE and UIS$PLOT were used in Tool Box. Since the application program called for menus of straight lines, there was no need to call UIS$CIRCLE or UIS$ELLIPSE.

1. UIS$LINE (VD_ID,ATTRIB,X1,Y1,X2,Y2, ... Xn,Yn)

This routine draws a point or a series of unconnected lines, depending upon the number of world coordinates which are used as arguments. The line style, color, and width are described in the attribute block. The default attribute block is block zero (0) which is modified using UIS$SET_LINE_STYLE() and UIS_SET_LINE_WIDTH() routines. [Ref. 8:p 18-210]

- VD_ID is the virtual display name.
- ATTRIB is an integer which identifies the attribute block.
- X1,Y1, ... are world coordinate pairs (up to 252 pairs) of points to be connected with lines.

2. UIS$PLOT (VD-ID,ATTRIB,X1,Y1,X2,Y2,...XN,YN)

The plot routine draws a point, line or polygon using a series of points specified in the argument. The line width and style are stored in the attribute block and are changed the same as plot line. A maximum of 126 world coordinate pairs can be used in the argument. [Ref. 8:p 18-229]

3. UIS$ERASE (VD_ID,X1,Y1,X2,Y2)

Whenever a section of the screen needs to be cleared (as when changing a displayed instruction), the erase command will delete any text or graphic within a box, with lower left hand corner world coordinates of X1, Y1 and upper right corner coordinates of X2, Y2. [Ref. 8:p 18-73]

E. CREATING TEXT ON THE VIRTUAL DISPLAY.

The basic unit for text drawing to the screen is the character cell. It has a specified size and relation to other text character cells on the display. The size and spacing of text characters are independent of the size of the virtual display or window. When the display window size is changed or altered, text remains unchanged and will not distort as the graphics figures will.

As discussed previously, this is the reason that a virtual display should initially have an added factor of size for unexpected growth.

The text characters are drawn in accordance with the specifications contained in the active font. In order for the text to change characteristics, the appropriate attribute must be changed. The default attribute settings are contained in the zero (0) attribute block. This attribute block is never changed in itself, but copies of it can be altered and stored in user numbered (indexed) attribute blocks. [Ref. 8:p 9-1] The new attribute blocks can then be further modified and given a new index number or stored under the same number.

Text attribute routines exist for changing character rotation, character size, character slant, character spacing, font, margins,path, slope, mode and index. In Tool Box, the text size was changed and reverse video was used to emphasize portions of the screen and to highlight choices. The text routines used in the program follow.

1. UIS$SET_WRITING_MODE (VD_ID, ATT1,ATT2,MODE)

In general, the writing mode in attribute block ATT1 is modified to one of 14 available writing modes and stored as ATT2. The writing modes associated with an index control how graphics and text routines use the foreground and background when displaying objects or text. [Ref. 8:p 18-363]

- VD_ID is the name of the virtual display.
- ATT1 is the index of the attribute block to be modified.
- ATT2 is the index number assigned to the new modified attribute block. It is referenced in UIS$ calls for graphics and text
- MODE is one of 14 writing modes.

2. UIS$SET_CHAR_SIZE (VD_ID,ATT1,ATT2, ,XSCALE,YSCALE)

This routine changes the size of the characters in attribute block ATT1 to width of XSCALE (absolute coordinate scale factor) and height of YSCALE. The result is stored as attribute block ATT2. [Ref. 8:p 18-267]

3. UIS$TEXT (VD_ID,ATT,TEXT_STRING,X,Y)

UIS$TEXT is the basic text routine for drawing text to the virtual display at position X, Y. [Ref. 8:p 18-372]

- VD_ID is the name of the virtual display.
- ATT is the attribute block index number.
- TEXT_STRING is a 'character string' enclosed in single quotes or a character variable which has been assigned a character string.
- X,Y is the world coordinate pair at which the text begins being drawn.

F. KEYBOARD ROUTINES AND COMMUNICATION

All of the normal keyboard functions are disabled whenever a user created graphics window is the active display on the screen. If the application program is to be interactive with the keyboard, the program must contain routines for creating and enabling the keyboard. In addition, the program must contain routines for reading, storing, and decoding the keystrokes. If other than character strings are needed, the program must also have routines to convert from character string to either integer or real data entries. Such amenities as backspace, keystroke echo, scrolling and returns are also disabled and must be programmed into the application.

In order for the keyboard to become active, a virtual keyboard must be created. Then the virtual keyboard is associated with a particular viewport; and,

finally, it must be enabled. The application must then be able to read keyboard entries as described above.

1. **KB_ID = UIS$CREATE_KB ('SYS$WORKSTATION')**

This routine creates a virtual keyboard which is identified as KB_ID. KB_ID is a returned longword value and uniquely identifies the keyboard. It is used as an argument in calling routines requiring a keyboard application. Once created, the virtual keyboard can be connected to any number of viewports. [Ref. 8:p 18-28]

2. **UIS$ENABLE_KB (KB_ID,WD_ID)**

Enable Keyboard connects the physical keyboard to the virtual keyboard and associates them with a specific window. The KB icon in the upper right hand corner of the window will highlight indicating the keyboard is active for that window. [Ref. 8:p 18-68]

3. **UIS$DISABLE_KB (KB_ID)**

This routine disconnects KB_ID from the physical keyboard and disables communication between the keyboard and the window. The KB icon in the upper right corner of the window will be blank. [Ref. 8:p 18-58]

4. **UIS$DELETE_KB (KB_ID)**

The delete keyboard routine deletes the virtual keyboard, KB_ID. [Ref. 8:p 18-49]

G. ASYNCHRONOUS SYSTEM TRAP ROUTINES (AST)

Tool Box relies on asynchronous system trap routines to move from menu to menu and to chose options within a menu. It also uses ASTs to detect the pointer position and highlight choices when the pointer is within the selection range of the option.

An asynchronous system trap routine is a routine which specifies a certain set of run time events which will cause the execution of a defined routine or routines. After completion of the AST, control returns to the point of interruption.

Once the conditions for AST execution are defined, the routine will activate any time during program execution when the conditions are satisfied. The application program continuously monitors for these conditions while executing the main program.

Normally, a set of ASTs are defined and the operation of the system is placed in a waiting mode. The waiting mode is activated by CALL SYS$HIBER (). In order to activate the system again, CALL SYS$WAKE () is used to disable the hibernation.

All names of all subroutines used in the ASTs must be declared in an EXTERNAL statement, since the subroutine name is passed as an argument in a CALL statement. The EXTERNAL statement must be included before any executable statement in the calling program.

1. CALL UIS$SET_BUTTON_AST (VD_ID,WD_ID,ASTPRM ,,,X1,Y1,X2,Y2)

When the first button on the mouse is depressed and the pointer is within a trigger rectangle, the subroutine ASTPRM is executed. Note that there is a triple comma position in the argument list. These are space holders. [Ref. 8:p 18-260]

The first space holder can contain the address of a data structure to pass data to the ASTPRM. The second space is for a keybuffer which can receive button information. Since all data was passed using named COMMON statements and no button information was needed these spaces were left blank. The space holders must be used to prevent hard to find errors.

- VD_ID is the virtual display identifier.
- WD_ID is the viewport window identifier.
- ASTPRM is the name of a subroutine to be executed when the button is depressed and the pointer is within a defined trigger rectangle.
- X1,Y1 is the lower left corner world coordinate pair.
- X2,Y2 is the upper right corner of the trigger rectangle.

2. UIS$SET_POINTER_AST (VD_ID,WD_ID,INAST, ,X1,Y1,X2,Y2,OUTAST)

The pointer AST monitors the world coordinates of the pointer and when it is within a specified box, the entering subroutine is executed. When the pointer exits the box, the exit AST is executed. [Ref. 8:p 18-328]

- VD_ID is the virtual display identifier.

- WD_ID is the viewport window identifier.
- INAST is the name of the subroutine to be executed when the pointer position crosses into the rectangle.
- X1,Y1 is the world coordinate of the lower left corner of the trigger rectangle.
- X2,Y2 is the upper right corner of the rectangle.
- OUTAST is the name of the subroutine to be executed upon pointer crossing out of the rectangle.

Neither routine needs to be specified, but the space holding commas need to be retained to prevent errors. The ASTs can be disabled by calling the routine again, leaving out the INAST and OUTAST names.

H. MISCELLANEOUS CALLABLE ROUTINES.

1. UIS$GET_POINTER_POSITION(VD_ID,WD_ID,X,Y)

- VD_ID is the virtual display identifier.
- WD_ID is the viewport window identifier.
- X,Y is the world coordinate position of the pointer

This routine is used to report the position of the pointer when AST routines are executed. The coordinates set up highlight and text changes to reflect the AST actions. [Ref. 8:p 18-160]

2. UIS$SET_POINTER_POSITION(VD_ID,WD_ID,X,Y)

- VD_ID is the virtual display identifier.
- WD_ID is the viewport window identifier.

- X,Y is the new world coordinate pointer position.

This routine is used to move the pointer out of the menu trigger field after a button AST choice has been made. By moving the pointer well out of the field as soon as the button is depressed, double button pushes (bounces) are avoided and the AST only activates one time. [Ref. 8:p 18-335]

3. CALL LIB$SPAWN('DCL_COMMAND')

This allows running nearly all Digital Command Language commands from within the application without leaving. This routine forms the basis of the Utilities option of the main menu. It is an extremely powerful and flexible routine, as the DCL_COMMAND can be in the form of a variable. As a variable, a character string representing any DCL command may be assigned to DCL_COMMAND to run virtually any option. [Ref. 4:p 3-2]

4. INCLUDE (FILENAME.FILETYPE)

During compilation of the program, whenever the INCLUDE 'FILENAME.FILETYPE' is reached, the compiler loads the contents of filename.filetype into the program at the location of the INCLUDE statement. The file must contain executable statements but it exists as a completely separate file. [Ref. 4:p 4-22]

Using this method, a common file containing NAMED COMMON, REAL and CHARACTER declarations etc., can be written for several related subroutines and "included" eliminating potential errors and saving programming time. Sections which contain long lists of assignment statements or similar blocks which detract from readability can be written as separate files and "included" at compile time.

CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The incorporation of the UIS graphics routines into a FORTRAN program produced an easy to use program which provides a solution for the desired problem. The ability to use the mouse and windows (to direct the program) works very well and allows the operator to concentrate on the problem. The windows approach also encourages modularity.

It was found that adding more displayed windows to the screen slowed the response time of the system. It was for this reason that in only one case are two Tool Box windows active and displayed at the same time.

It was further discovered that the addition of more active ASTs reduced the response time of the system. By limiting the active ASTs to those actually needed by the controlling menu, response times can be kept quite reasonable.

A program written using the UIS graphics routines can not be transported to any other system which is not supported with the UIS libraries. If transportability is a requirement, some of the new, compilable, BASIC languages, such as QUICK BASIC or TURBO BASIC, could be used to achieve similar results.

B. RECOMMENDATIONS

Tool Box starts the problem well into the preliminary design phase. Several sections should be produced to generate the baseline input data which was assumed to be available for static stability and maneuvering performance calculations.

One such section could read in hull form information by integrating a hull form sketch drawn to screen by the pointer. Similar techniques could provide wetted areas, moments of inertia, power and resistance estimates or offset data.

Tool Box used only a few of the available UIS routines and the results were very good. There are still many capabilities provided which need to be explored. More applications should be written, with an eye toward building an in house knowledge base of the use of these graphics capabilities.

APPENDIX A

MANEUVERING DYNAMICS REVIEW

The equations of motion are well established and only a very general treatment is used here as a point of reference. This review is condensed from Clarke, Gedling and Hine, [Ref 6].

The motion of a vessel is usually a six degree of freedom system, but only the equations of surge, sway, and yaw are treated here. Using the standard co-ordinate systems, the equations of motion are found to be

$$\begin{aligned} X &= m(\dot{u} - rv - x_g r^2) \\ Y &= m(\dot{v} + ur + x_g \dot{r}) \\ N &= I_z \dot{r} + m x_g (\dot{v} + ru) \end{aligned} \qquad (1)$$

(hydrodynamic forces = inertial response)

The hydrodynamic forces are usually expressed as linear perturbations about a fixed steady state forward speed u_0, and are expressed as

$$\begin{aligned} X &= X_{\dot{u}} \dot{u} + X_u u \\ Y &= Y_{\dot{v}} \dot{v} + Y_v v + Y_{\dot{r}} \dot{r} + Y_r r \\ N &= N_{\dot{v}} \dot{v} + N_v v + N_{\dot{r}} \dot{r} + N_r r \end{aligned} \qquad (2)$$

The sway and yaw equations from (1) and (2) are combined and a term is added for the rudder (δ). Keeping only the linear terms and non-dimensionalizing the resulting equations are:

$$(Y'_{\dot{v}} - m)\dot{v}' + Y'_{v}\, v' + (Y'_{\dot{r}} - m'x_g)\dot{r}' + (Y'_{r} - m')r + Y'_{\delta}\,\delta = 0 \qquad (3)$$

$$(N'_{\dot{v}} - m'x_g)\dot{v}' + N'_{v}v' + (N'_{\dot{r}} - I'_z)\dot{r}' + (N'_{r} - m'x'_g)r' + N'_{\delta}\delta = 0$$

Several approaches have ben used to estimate the derivatives in (3) including strip theory, semi-empirical methods and multiple regression. The obvious aims of all of the studies has been to correlate overall hull geometry to hydrodynamic derivatives.

Looking at the ship, it must be immediately obvious that there are a few parameters which must have large effect on the maneuverability and stability. Since they must have a large effect the hydrodynamic forces and responses, these must be included in at least the initial fit of data. The parameters are length between perpendiculars (L), draft (T), block coefficient (C_B) and Beam (B).

There are also some parameters, not immediately obvious, which also have an effect on the correlations. These are stern type, propeller direction of rotation and number,

rudder type (span, chord, area, and thickness), number of rudders, and bow area.

In doing the multiple regression analysis, Clarke, Gedling, and Hine used 36 sets of data from rotating arm experiments and 36 sets of data from planar motion experiments. Each of the derivatives was normalized using - $(T/L)^2$. As predictor variables, they chose C_B, L/B, L/T, B/T, B/L, T/L, T/B, and their products and squares. Only the terms which tended to zero were used in the regression to assure stability.

For each derivative, the regression was first started giving the simple terms preference and then adding the higher order terms to check for significant improvement. The resulting coefficients were significant to the 5% level using t-statistics except N'_v which was only .6% significant. The estimating model from multiple linear regression is

$$
\begin{aligned}
Y'_{\dot{v}} &= - (T/L)^2 (1 + .16\, C_B B/T - 5.1 (B/L)^2) \\
Y'_{\dot{r}} &= - (T/L)^2 (.67\, B/L - .041\, B/T) \\
Y'_{v} &= - (T/L)^2 (1 + .4\, C_B B/T) \\
Y'_{r} &= - (T/L)^2 (-1/2 + 2.2\, B/L - .08\, B/T) \\
N'_{\dot{v}} &= - (T/L)^2 (1.1\, B/L - .041\, B/T) \\
N'_{\dot{r}} &= - (T/L)^2 (1/12 + .017\, C_B B/T - .33\, B/L) \\
N'_{v} &= - (T/L)^2 (1/2 + 2.4\, T/L) \\
N'_{r} &= - (T/L)^2 (1/4 + .039\, B/T - .56\, B/L
\end{aligned}
\qquad (4)
$$

When a ship is being designed, the parameters available to the architect are physical quantities which he has set down to describe and communicate the vessel. He can not easily design using concepts of $N_{\dot{v}}$ or Y_r .

With the above correlations (4), and the linear equations of motion (3), an attempt at describing the performance of the vessel can be undertaken. Nomoto, [Ref. 9] showed that equations (3) could be expressed as a set of decoupled, second order differential equations

$$T_1' T_2' \ddot{r}' + (T_1' + T_2') \dot{r} + r' = K'\delta + T_3' \delta'$$

$$T_1' T_2' \ddot{v}' + (T_1' + T_2') \dot{v} + v' = K_v' \delta + K_v' T_3' \delta' \qquad (5)$$

The rudder derivatives, rudder fin stabilizing effect and a water depth correction, outlined in Appendix 1 and Appendix 2 of Reference 4 was also incorporated.

This set of equations can then be solved in terms of the hydrodynamic derivatives with results as follows.

$$T_1'T_2' = \frac{(Y'_{\dot{v}} - m')(N'_{\dot{r}} - I'_z) - (Y'_{\dot{r}} - m'x'_g)(N'_{\dot{v}} - m'x'_g)}{Y'_v(N'_r - m'x'_g) - N'_v(Y'_r - m')}$$

$$T'_1+T'_2 =$$

$$\frac{(Y'_{\dot{v}}-m')(N'_r-m'x'_g)+(N'_{\dot{r}}-I'_z)Y'_v-(Y'_r-m'x'_g)N'_v-(N'_{\dot{v}}-m'x'_g)(Y'_r-m')}{Y'_v(N'_r-m'x'_g)-N'_v(Y'_r-m')}$$

$$T'_3 = \frac{(N'_{\dot{v}}-m'x'_g)Y'_\delta-(Y'_{\dot{v}}-m')N'_\delta}{N'_v Y'_\delta - Y'_v N'_\delta}$$

$$T'_4 = \frac{(N'_{\dot{r}}-I'_z)Y'_\delta-Y'_{\dot{r}}-m'x'_g)N'_\delta}{(N'_r-m'x'_g)Y'_\delta-(Y'_r-m')N'_\delta}$$

$$K' = \frac{N'_v Y'_\delta - Y'_v N'_\delta}{Y'_v(N'_r-m'x'_g)-N'_v(Y'_r-m')}$$

$$-K'_v = \frac{(N'_{\dot{v}}-m'x'_g)Y'_\delta-(Y'_{\dot{v}}-m')N'_\delta}{Y'_v(N'_r-m'x'_g)-N'_v(Y'_r-m')}$$

$$T' = T'_1 + T'_2 + T'_3$$

Clarke, Gedling and Hine [Ref. 6], solve the above relationships for a heading response to a given rudder input and call it the Turning Index, P. P is the ratio of heading response to rudder angle and a ratio of $P > 0.3$ is considered adequate. For a given rudder angle and non-dimensional time of 1 (one ship length moved) the turning index can be represented as:

$$P = K' \left[1 - (T'_1 + T'_2 - T'_3) + \frac{(T'_1 - T'_3)}{(T'_1 - T'_2)} T'_1 e^{-1/T'_1} + \frac{(T'_2 - T'_3)}{(T'_1 - T'_2)} T'_2 e^{-1/T'_2} \right]$$

The ship is dynamically course stable if the time constants T'_1 and T'_2 are positive. This is true if

$$Y'_v(N'_r - m'x'_g) - N'_v (Y'_r - m') > 0 \quad .$$

C. A. Lyster and H. 1. Knights, [Ref. 7], also performed analysis on a large set of ship turning data in an attempt to formulate predicting equations for ships' turning circles. The results produced two sets of equations, single screw and twin screw ships which are presented below.

SINGLE SCREW

$$STD/L = 4.19 - 203\ Cb/\delta + 47.4\ TRIM/L - 13B/L + 194/\delta - 35.8\ SpCh(ST-1)/LT + 3.82SpCh(ST-2)/LT + 7.79Ab/LT + .7(T/T1 - 1)(\delta/|\delta|)(ST-1)$$

$$TD/L = .91\ STD/L + .424\ Va/\ L + .675$$

$$AD/L = .519\ TD/L + 1.33$$

$$TR/L = .497\ TD/L - .065$$

$$Vt/Va = .074\ TD/L + .149$$

APPENDIX B

STATIC STABILITY REPORT

TOOL BOX

INITIAL STABILITY REPORT

THIS REPORT WAS GENERATED USING THE PROGRAM

TOOL BOX WHICH WAS DEVELOPED FOR THE NAVAL

ENGINEERING DEPARTMENT OF THE

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

BY

PROFESSOR F. PAPOULIAS

AND

LT. GERALD MCGOWAN

1990

TWIN SCREW

$$STD/L = .727 - 197\ Cb/|\delta| + 4.65\ B/L + 41\ TRIM/L + 188\ /|\delta| - 218\ SpCh\ (Nr-1)/LT + 3.2\ Va/\sqrt{L} + 25.56\ Ab/LT$$

$$TD/L = .140 + STD/L$$

$$AD/L = 1.1 + .514\ TD/L$$

$$TR/L = -.357 + .531\ TD/L$$

$$Vt/Va = .543 + .028\ TD/L$$

Ab = Submerged bow area profile

AD = Advance at 90 degrees of turn.

B = Beam in feet

Cb = block coefficient.

Ch = Chord of the rudder in feet.

Sp = Span of rudder in feet.

ST = Stern type. (1 = closed, 2 = open).

STD = Steady turning diameter in feet.

T = Trial Draft in feet.

TD = Tactical Diameter in feet.

Tl = Design Draft in feet.

TR = Transfer at 90 degrees of turn.

Va = Approach velocity.

Vt/Va = Velocity drop in the steady turn.

Vt = Velocity on steady turn.

δ = Rudder Angle in degrees (Negative = stbd)

INITIAL STABILITY REPORT

THE INPUT SHIP PARAMETERS ARE AS FOLLOWS

THE REPORT IS LOCATED IN FILE DD692S.RPT
THE INPUT DATA FILE USED IS DD692S.DAT

SHIP NAME IS DD692S

LENGTH BETWEEN PERPENDICULARS	FT	383.00
LENGTH AT DESIGN WATERLINE	FT	383.00
DESIGN LOAD DRAFT	FT	13.000
MEAN TRIAL DRAFT	FT	13.000
MOULDED BEAM (MAX)	FT	40.600
DISPLACEMENT AT DESIGN LOAD	LTONS	3000.0
VOLUMETRIC DISPLACEMENT	FT**3	0.10500E+06
MAXIMUM CROSS SECTION AREA	FT**2	431.73
MIDSHIPS SECTION AREA	FT**2	422.24
WATERPLANE AREA AT DESIGN DRAFT	FT**2	11878.
LONGITUDINAL CENTER OF GRAVITY	Xg	-22.200
BLOCK COEFFICIENT		0.51942
PRISMATIC COEFFICIENT		0.63500
MIDSHIPS COEFFICIENT		0.80000
VERTICAL PRISMATIC COEFFICIENT		0.68000
WATERPLANE AREA COEFICIENT		0.76386
TRANS. WATERPLANE INERTIA COEFFICIENT		0.55962
LONG. WATERPLANE INERTIA COEFFICIENT		0.55432
TRANSVERSE SECOND MOMENT OF AREA		0.11953E+07
LONGITUDINAL SECOND MOMENT OF AREA		0.10537E+09
TONS PER INCH IMMERSION		28.281
MOMENT TO TRIM ONE INCH		650.00
GM	FT	3.7000

BM	FT	11.384
KM	FT	19.271
KG	FT	15.571
KB	FT	7.8867
LOGITUDINAL GM	FT	995.80
LOGITUDINAL BM	FT	1003.5
LOGITUDINAL KM		1011.4

TRANSVERSE AND VERTICAL WEIGHT SHIFT EFFECTS

NUMBER OF LON TONS OF WEIGHT ADDED	25.000
VERTICAL DISTANCE FROM CENTER OF GRAVIT	2.5000
TRANSVERSE DISTANCE FORM CENTERLINE	3.2000
NEW CENTER OF BOUYANCY	7.9314
NEW KG	15.591
NEW GM	3.6299
NEW KM	19.221
NEW BM	11.290
LIST ANGLE. NEGATIVE = STARBOARD	0.41743
FINAL DISPLACEMENT	3025.0
FINAL MEAN DRAFT	13.074

LONGITUDINAL WEIGHT SHIFT EFFECTS

WEIGHT ADDED IN LONG TONS	22.000
LONGITUDINAL DISTANCE TO MIDSHIPS	14.000
LONGITUDINAL CENTER OF FLOATATION	-22.200
FINAL DISPLACEMENT IN LONG TONS	3022.0
CHANGE IN TRIM	-1.2252
DRAFT AFT	12.523
DRAFT FORWARD	13.748
NEW MEAN DRAFT	13.136

TRIM ANGLE .....-0.18329

ITERATIONS OF TRANSVERSE STABILITY
WITH GM HELD CONSTANT AND TSI VARYING

TSI		0.19000
GM	FT	3.7000
BM	FT	11.587
KM	FT	19.474
KG	FT	15.774
KB	FT	7.8867

TSI		0.19200
GM	FT	3.7000
BM	FT	11.384
KM	FT	19.271
KG	FT	15.571
KB	FT	7.8867

TSI		0.19400
GM	FT	3.7000
BM	FT	11.185
KM	FT	19.072
KG	FT	15.372
KB	FT	7.8867

APPENDIX C

MANEUVERING PERFORMANCE REPORT

TOOL BOX

MANEUVERING PERFORMANCE REPORT

THIS REPORT WAS GENERATED USING THE PROGRAM

TOOL BOX WHICH WAS DEVELOPED FOR THE NAVAL

ENGINEERING DEPARTMENT OF THE

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

BY

PROFESSOR F. PAPOULIAS

AND

LT. GERALD MCGOWAN

1990

TOOL BOX REPORT

THE INPUT SHIP PARAMETERS ARE AS FOLLOWS

THE REPORT IS LOCATED IN FILE	DD692M.RPT
THE INPUT DATA FILE USED IS	DD692M.DAT
SHIP NAME IS	DD692M
SHIP LENGTH.....(FEET).....	383.00
WATER LINE MAX BEAM .....(FEET).....	40.600
SHIP DESIGN DRAFT.....(FEET).....	13.000
SHIP DISPLACEMENT..... (SW).(L TONS).....	3000.0
BLOCK COEFICIENT	0.51942
CENTER OF GRAVITY .(LONG. NEG = AFT).....	-22.200
TRIAL TRIM OF SHIP.....(FEET)	0.00000E+00
ACTUAL TRIAL DRAFT.....(FEET)	13.000
SPEED OF APPROACH .....(KNOTS)	25.000
SUBMERGED PROFILE BOW AREA ...(FT**2).....	25.000
NUMBER OF RUDDERS	1
RUDDER AREA .....(FT**2 PER RUDDER).....	124.50
RUDDER ANGLE ....(DEG. NEG = STBD).....	30.000
STERN TYPE...(1 = CLOSED, 2 = OPEN)	1
NUMBER OF SCREWS.....	2
DEPTH OF WATER .....(FEET).....	10000.

PAGE TWO OF THIS REPORT CONTAINS THE PREDICTED TURNING CIRCLE PERFORMANCE BASED UPON THE DATA ABOVE.

PAGES 3 AND 4 CONTAIN THE CALCULATEDHYDRODYNAMIC DERIVATIVES AND THE MANEUVERING PERFORMANCE.

PAGES 5 AND 6 ARE GENERATED IF USER SUPPLIEDHYDRODYNAMIC DERIVATIVES WERE USED AND STORED WITH THE DATA.

TOOL BOX REPORT

DD692M.RPT

THE TURNING CIRCLE DATA PRESENTED HERE
ARE BASED UPON PREDICTION EQUATIONS PRESENTED BY
C. A. LYSTER AND MRS. H. L. KNIGHTS IN

"PREDICTION EQUATIONS FOR SHIPS' TURNING CIRCLES"
NORTHEAST COAST INSTITUTION OF ENGINEERS AND SHIPBUILDERS
TRANSACTIONS, VOLUME 95, NO. 4

THE SHIP TURNING CHARACTERISTICS ARE PREDICTED AS:

STEADY TURNING DIAMETER.....(FEET).....	3175.8
TACTICAL DIAMETER .....(FEET).....	3229.4
ADVANCE AT 90 DEGREES OF TURN .(FEET).....	2081.2
TRANSFER AT 90 DEGREES .....(FEET).....	1578.1
STEADY TURN VELOCITY .....(KNOTS).....	19.477

THE NON DIMENSIONAL TURNING CIRCLE FACTORS ARE:

STD'	8.2919
TR'	8.4319
AD'	5.4340
TR'	4.1203
VT/VA	0.77909

TOOL BOX REPORT

DD692M.RPT

MANEUVERING PERFORMANCE USING CALCULATED
HYDRODYNAMIC DERIVATIVES FROM INPUT SHIP DATA

BASED UPON NUMERICAL MODELS PROPOSED BY

D. CLARK, P. GEDLING AND G. HINE AS FOUND IN

"THE APPLICATION OF MANEUVERING CRITERIA

IN HULL DESIGN USING LINEAR THEORY"

THE NAVAL ARCHITECT, MARCH, 1983.

THE LINEAR HYDRODYNAMIC DERIVATIVES:

M'	0.37378E-02
Iz'	0.23361E-03
Y'v DOT	-0.43517E-02
Y'r DOT	-0.14057E-03
N'v DOT	0.41408E-04
N'r DOT	-0.27482E-03
Y'v	-0.67330E-02
Y'r	0.22520E-02
N'v	-0.17227E-02
N'r	-0.13218E-02
Y'del	0.25462E-02
N'del	-0.12731E-02

TOOL BOX REPORT

DD692M.RPT

THE RESULTING PERFORMANCE FACTORS:

DOMINANT SHIP TIME CONSTANT T1' 2.2686

SHIP TIME CONSTANT ..... T2' 0.36964

TIME CONSTANT ..... T3' 0.74406

TIME CONSTANT ..... T4' 0.25452

NOMOTTO FIRST ORDER TIME C . T' 1.8942

RUDDER LOOP GAIN FACTOR .... K' ..... -2.6546

RUDDER GAIN FACTOR ..... K'v .....-0.96400

STEERING GEAR TIME CONSTANT T'E ..... 0.00000E+00

|1/T'| 0.52794

|1/K'| 0.37670

TURNING PERFORMANCE INDEX .. (P) 0.75056

DYNAMIC STABILITY INDEX ... (C) 0.48814E-05

SHIP IS DYNAMICALLY COURSE STABLE

********* EQUATIONS OF STATE **********

$$\frac{d}{dT}\begin{bmatrix}\psi\\ r\\ v\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\ 0 & A22 & A23\\ 0 & A32 & A33\end{bmatrix}\begin{bmatrix}\psi\\ r\\ v\end{bmatrix} + \begin{bmatrix}0\\ B2\\ B3\end{bmatrix} DEL + \begin{bmatrix}0 & 0\\ G21 & G22\\ G31 & G32\end{bmatrix}\begin{bmatrix}Y\ EXT\\ N\ EXT\end{bmatrix}$$

A22 = -2.2778 A23 = -3.8290

A32 = -0.20510 A33 = -0.86833

B2 = -2.3555 B3 = 0.29260

G21 = 63.046 G22 = 1976.3

G31 = 124.21 G32 = 18.589

TOOL BOX REPORT

DD692M.RPT

MANEUVERING PERFORMANCE WITH USER SUPPLIED
HYDRODYNAMIC DERIVATIVES

BASED UPON NUMERICAL MODELS PROPOSED BY

D. CLARK, P. GEDLING AND G. HINE AS FOUND IN

"THE APPLICATION OF MANEUVERING CRITERIA

IN HULL DESIGN USING LINEAR THEORY"

THE NAVAL ARCHITECT, MARCH, 1983.

THE LINEAR HYDRODYNAMIC DERIVATIVES:

M'	0.37378E-02
Iz'	0.23361E-03
Y'v DOT	-0.43517E-02
Y'r DOT	-0.14057E-03
N'v DOT	0.41408E-04
N'r DOT	-0.27482E-03
Y'v	-0.67330E-02
Y'r	0.22520E-02
N'v	-0.17227E-02
N'r	-0.13218E-02
Y'del	0.25462E-02
N'del	-0.13000E-02

TOOL BOX REPORT

DD692M.RPT

THE RESULTING PERFORMANCE FACTORS:

DOMINANT SHIP TIME CONSTANT T1' 2.2686

SHIP TIME CONSTANT ..... T2' 0.36965

TIME CONSTANT ..... T3' 0.75037

TIME CONSTANT ..... T4' 0.25196

NOMOTTO FIRST ORDER TIME C . T' 1.8879

RUDDER LOOP GAIN FACTOR .... K' ..... -2.6918

RUDDER GAIN FACTOR ..... K'v .....-0.97219

STEERING GEAR TIME CONSTANT T'E ..... 0.00000E+00

|1/T'| 0.52970

|1/K'| 0.37150

TURNING PERFORMANCE INDEX .. (P) 0.76520

DYNAMIC STABILITY INDEX ... (C) 0.48813E-05

SHIP IS DYNAMICALLY COURSE STABLE

********* EQUATIONS OF STATE **********

$$\frac{d}{dT}\begin{vmatrix}\psi\\ R\\ V\end{vmatrix} = \begin{vmatrix}0 & 1 & 0\\ 0 & A22 & A23\\ 0 & A32 & A33\end{vmatrix}\begin{vmatrix}\psi\\ R\\ V\end{vmatrix} + \begin{vmatrix}0\\ B2\\ B3\end{vmatrix}\delta_R + \begin{vmatrix}0 & 0\\ G21 & G22\\ G31 & G32\end{vmatrix}\begin{vmatrix}Y_{EXT}\\ N_{EXT}\end{vmatrix}$$

A22 = -2.2777 A23 = -3.8290

A32 = -0.20510 A33 = -0.86832

B2 = -2.4086 B3 = 0.29210

G21 = 63.045 G22 = 1976.3

G31 = 124.21 G32 = 18.588

APPENDIX D

TOOL BOX WORKSHEET

STATIC STABILITY

1. SHIP NAME .....
2. SHIP LENGTH .....(LPP FEET).
3. SHIP LENGTH .....(LWL FEET).
4. DESIGN DRAFT .....(FEET)...
5. MEAN DRAFT .....(FEET)...
6. MAXIMUM BEAM .....(FEET)...
7. DESIGN DISPLACEMENT .....(LONG TONS).
8. BLOCK COEFFICIENT .....(Cb)....
9. PRISMATIC COEFFICIENT .....(Cp)....
10. MIDSHIPS COEFFICIENT .....(Cm)....
11. VERTICAL PRISMATIC COEFFICIENT .....(Cp)....
12. LONGITUDINAL CENTER OF GRAVITY .....(Xg)....

TRANSVERSE STABILITY

* 13. TRANSVERSE STABILITY INDEX .....(GM/KM) .
* 14. KG .....(FEET)...
* 15. GM .....(FEET)..
* 16. KM .....(FEET)...
* 17. BM .....(FEET)...
18. TONS OF WEIGHT ADDED .....(LONG TONS).
19. VERTICAL DISTANCE WEIGHT TO CL .....(FEET)...
20. TRANSVERSE DISTANCE WEIGHT TO CL .....(FEET)...

LONGITUDINAL STABILITY

**21. LONGITUDINAL GM .....(FEET)...
**22. LONGITUDINAL BM .....(FEET)...
**23. LONGITUDINAL KM .....(FEET)...
**24. LONGITUDINAL MOMENT .....(Il).....
**25. LONGITUDINAL WATER[LANE COEFFICIENT ...(Cil)....
**26. MOMENT TO TRIM ONE INCH .....(MTl).....
27. WEIGHT ADDED IN LONG TONS.....
28. LONGITUDINAL DISTANCE WEIGHT TO MIDSHIPS.....
29. LONGITUDINAL CENTER OF FLOATATION (NEG. = AFT).

* User must furnish any two of these.
** User must furnish any one of these.

MANEUVERING PERFORMANCE

1. SHIP NAME ..... _____
2. SHIP LENGTH .....(LPP FEET). _____
3. SHIP BEAM .....(FEET)... _____
4. DESIGN DRAFT .....(FEET)... _____
5. DESIGN DISPLACEMENT .....(FEET)... _____
6. BLOCK COEFFICIENT .....(Cb).... _____
7. CENTER OF GRAVITY .....(Xg).... _____
8. TRIM .....(FEET)... _____
9. TRIAL DRAFT .....(FEET)... _____
10. SPEED OF ADVANCE INTO TURN .....(KNOTS).. _____
11. SUBMERGED BOW AREA .....(SQ. FEET). _____
12. NUMBER OF RUDDERS ..... _____
13. TOTAL RUDDER AREA .....(SQ. FEET). _____
14. RUDDER ANGEL IN TURN .....(DEG)... _____
15. STERN TYPE (1 = CLOSED, 2 = OPEN)..... _____
16. NUMBER OF SCREWS ..... _____
17. WATER DEPTH .....(FEET)... _____

USER SUPPLIED HYDRODYNAMIC COEFFICIENTS (NOT REQUIRED)

18. _____ ..... _____
19. _____ ..... _____
20. _____ ..... _____
21. _____ ..... _____
22. _____ ..... _____
23. _____ ..... _____
24. _____ ..... _____
25. _____ ..... _____
26. _____ ..... _____
27. _____ ..... _____

APPENDIX E

TOOL BOX MAIN PROGRAM

```
          PROGRAM TB
************************************************************************
*     THE TOOL BOX MAIN PROGRAM ESTABLISHES THE VIRTUAL DISPLAY
* BOUNDARIES AND DRAWS THE WORKING GRAPHICS AND TEXT TO IT.  IT THEN
* CREATES RUN TIME EVENT TRAPS FOR HIGHLIGHTING CHOICES AND FOR
* CHOICE SELECTION.  FINALLY, IT GOES INTO HIBERNATION AWAITING THE
* RUN TIME EVENTS TO TRIGGER SUBSEQUENT SUBROUTINES.
************************************************************************
      INCLUDE 'GENERAL.FOR/LIST'
      REAL DY2

**** SINCE THE SUBROUTINES ARE PASSED AS ARGUMENTS IN THE AST ROUTINES
**** THEY MUST BE CALLED IN EXTERNAL STATEMENTS
      EXTERNAL NOWHERE, REPORT
      EXTERNAL DARK,DARK2,LIGHT,STATIC,EXIT,MANEUVER, UTIL


**** CREATE A NEW VIRTUAL DISPLAY ************************************
        VD_ID=UIS$CREATE_DISPLAY(-1.,-1.,20.,25.5,40.0,30.0)

**** AND ADD COLORS TO IT ********************************************
      CALL UIS$SET_COLOR(VD_ID,1,.75,1.,1.)
      CALL UIS$SET_COLOR(VD_ID,0,0.,0.,.5)

**** SET UP SOME CHARACTER TYPES**************************************
       CALL UIS$SET_WRITING_MODE(VD_ID,0,1,UIS$C_MODE_REPLN)
       CALL UIS$SET_WRITING_MODE(VD_ID,1,2,UIS$C_MODE_REPL)
       CALL UIS$SET_WRITING_MODE(VD_ID,0,4,UIS$C_MODE_REPLN)
       CALL UIS$SET_CHAR_SIZE(VD_ID,4,5,,.138,.39)
       CALL UIS$SET_CHAR_SIZE(VD_ID,0,7,,.125,.3)
```

```
*************INITIALIZE*************************************************
       X0 = 0.
       X1 = 3.0
       Y_LINE = .7
       FL1 = .FALSE.
       FL2 = .FALSE.
       YPOS = 1.

***************DRAW THE MAIN MENU*****************************************
************************************************************************
C  THIS ROUTINE DRAWS THE TITLE BOXES FOR THE MAIN MENU
       DO 10 Y_COOR = 0.2,5.8,.8
          DY = Y_COOR + .4
          CALL UIS$PLOT(VD_ID,0,0.,Y_COOR,3.,Y_COOR,3.,DY,0.,
     &          DY,0.,Y_COOR)
10     CONTINUE

**** CALL UP THE OPTIONS TITLES FILE ********************************
       INCLUDE 'OPTIONS.FOR'

**** NOW FILL THE BLOCKS OF THE MAIN MENU WITH THE STORED TITLES *****
       DO 20 I = 1, 8
          CALL UIS$TEXT(VD_ID,0,OPTION(I),.3,Y_LINE)
          Y_LINE = Y_LINE + .8
20     CONTINUE


**** TURN ON THE MAIN MENU DISPLAY VIEWPORT **************************
        WD_MAIN=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','MAIN MENU',
     &  -.5,-.5,10.,7.,40.,30.)

****************** SET UP THE AST'S FOR HIGHLIGHTING *****************
      DO 50 Y_COOR = .2,5.8,.8
         DY = Y_COOR + .2
         CALL UIS$SET_POINTER_AST(VD_ID,WD_MAIN,DARK,,X0,Y_COOR,X1,
     &    DY,LIGHT)
50    CONTINUE
```

```
******SET MAIN MENU TRAPS*************************************************
*****************************************************************************
        CALL UIS$SET_BUTTON_AST(VD_ID,WD_MAIN,EXIT,,,X0,.2,X1,.6)
        CALL UIS$SET_BUTTON_AST(VD_ID,WD_MAIN,REPORT,,,X0,1.0,X1,1.4)
        CALL UIS$SET_BUTTON_AST(VD_ID,WD_MAIN,UTIL,,,X0,1.8,X1,2.2)
        CALL UIS$SET_BUTTON_AST(VD_ID,WD_MAIN,NOWHERE,,,X0,2.6,X1,3.0)
        CALL UIS$SET_BUTTON_AST(VD_ID,WD_MAIN,NOWHERE,,,X0,3.4,X1,3.8)
        CALL UIS$SET_BUTTON_AST(VD_ID,WD_MAIN,NOWHERE,,,X0,4.2,X1,4.6)
        CALL UIS$SET_BUTTON_AST(VD_ID,WD_MAIN,MANEUVER,,,X0,5.0,X1,5.4)
        CALL UIS$SET_BUTTON_AST(VD_ID,WD_MAIN,STATIC,,,X0,5.8,X1,6.6)

**** PUT IN THE TITLE BLOCK TEXT ***************************************
        INCLUDE 'HEADER.FOR/LIST'

*** DRAW THE STATIC / MANEUVERING INTERACTIVE DISPLAY*****************
*****************************************************************************
        CALL UIS$PLOT(VD_ID,0,-.5,10.,10.,10.,10.,18.8,-.5,
     &                18.8,-.5,10.)
        CALL UIS$LINE(VD_ID,0,3.5,10.,3.5,18.8,-.5,11.,3.5,11.,
     &    -.5,11.4,3.5,11.4,-.5,12.4,3.5,12.4,-.5,12.8,3.5,12.8,
     &    -.5,18.4,10.,18.4)

        DO 30 Y_COOR = 13. , 18.4, .8
           DY = Y_COOR + .4
           DY2 = DY +.1
           Z = 9 + INT((Y_COOR - 12.9)/.8)
           CALL UIS$TEXT(VD_ID,0,OPTION(Z),.3,DY2)
           CALL UIS$PLOT(VD_ID,0,X0,Y_COOR,X1,Y_COOR,X1,DY,X0,
     &     DY,X0,Y_COOR)
30      CONTINUE

**** FILL THE TITLE BLOCKS OF WORKING DISPLAY ************************
        CALL UIS$TEXT(VD_ID,5,BLOCK(1),-.5,12.85)
        CALL UIS$TEXT(VD_ID,5,BLOCK(2),-.5,11.4)
        CALL UIS$TEXT(VD_ID,5,BLOCK(3),-.5,18.8)
        CALL UIS$TEXT(VD_ID,5,LONG_TITLE(1),4.,18.8)
        CALL HCUIS$WRITE_DISPLAY(VD_ID,'HARD.UIS')
```

```
**** NOW DRAW UP THE REPORT SCREEN ************************************
        CALL RPT_GRAPH

**** FINALLY, WAIT FOR THINGS TO HAPPEN ******************************
        CALL SYS$HIBER()

40      END
```

APPENDIX F

SUBROUTINE KEY_READ

```
**** START KEY READ SUBROUTINE*******************************
************************************************************

      SUBROUTINE KEY_READ (LINE,FCHAR,XSTART,YSTART,*)

      INCLUDE 'TOP.FOR'

      LOGICAL FCHAR, FPRIOD
      LOGICAL*4  KEY_IN
      LOGICAL*1 KEYBUF(4)
      EQUIVALENCE (KEY_IN,KEYBUF)
      REAL DEL_X,XSTART,YSTART,XSL,XSH,YSL,YSH
      CHARACTER  LINE*12, KEY_IS*1

**** FIND THE LOCATION OF THE KEY ECHO **********************
      XSL = XSTART - .1
      XSH = XSTART + 3.
      YSL = YSTART - .4
      YSH = YSTART + .1

**** START THE READ SEQUENCE ********************************
10    COUNT = 0
      FPRIOD = .TRUE.
11    KEY_IN=UIS$READ_CHAR(KB_ID)
      CALL LIB$CHAR(KEY_IS,KEYBUF(1))

**** IF BACKSPACE IS DETECTED, THE DATA NEEDS CHANGING ******
      IF ((COUNT .GT. 0) .AND. (KEYBUF(1) .EQ. 127)) THEN
         CALL UIS$ERASE(VD_ID,XSL,YSL,XSH,YSH)
         COUNT = COUNT - 1
         LINE = LINE(:COUNT)
         CALL UIS$TEXT(VD_ID,7,LINE,XSTART,YSTART)
         GO TO 11
      END IF

**** WHEN A PERIOD IS READ, SET THE FLAG ********************
        IF (KEYBUF(1) .EQ.46) FPRIOD = .FALSE.
```

```
**** IF [RETURN] IS DETECTED, TEST AND RETURN ******
       IF((KEYBUF(1) .EQ. 13) .AND. (COUNT .EQ. 0)) RETURN 1
       IF(KEYBUF(1) .EQ. 13) THEN
          CALL UIS$ERASE(VD_ID,XSL,YSL,XSH,YSH)
          IF(FPRIOD .AND. FCHAR) LINE = LINE(:COUNT)//'.'
          RETURN
       ELSE
          DEL_X = XSTART + .125 * COUNT
          LINE = LINE(:COUNT)//KEY_IS
          COUNT = COUNT + 1
          CALL UIS$TEXT(VD_ID,7,KEY_IS,DEL_X,YSTART)
          GO TO 11
       END IF

20     END
*******************************END KEY READ ***********
```

APPENDIX G

SUBROUTINE UTIL

```
************************************************************
*    UTILITY IS A SUBROUTINE WHICH ALLOWS THE USER         *
*  TO SHELL DCL COMMANDS WITHOUT LEAVING THE ENVIRONMENT   *
*  OF THE PROGRAM.  WHEN FINISHED WITH UTILITIES, ENTER    *
*  THE COMMAND 'EXIT' TO RETURN TO THE MAIN MENU ************

      SUBROUTINE UTIL

      INCLUDE 'GENERAL.FOR'

      CHARACTER *64  CMMD

      STA=UIS$SET_POINTER_POSITION(VD_ID,WD_MAIN,9.9,5.)

**** PUSH THE MAIN MENU TO THE BACK OF THE DISPLAY **********
      CALL UIS$PUSH_VIEWPORT(WD_MAIN)

**** WRITE THE INSTRUCTIONS UP TO THE SCREEN ****************
1     TYPE *, 'ENTER THE DCL COMMAND ...[RETURN] TO EXIT'
      PRINT *,' '
      READ (*,FMT='(BN,A)')  CMMD

**** ECHO THE COMMAND ***************************************
      PRINT *, CMMD
      IF (CMMD .EQ. ' ')  THEN
         CALL UIS$POP_VIEWPORT(WD_MAIN)
         RETURN
      ENDIF

**** PERFORM THE COMMAND ************************************
        CALL LIB$SPAWN(CMMD)

      PRINT *,' '
      GOTO 1

      END
*************************END OF SUBROUTINE UTIL *************
```

APPENDIX H

SUBROUTINE DATA_IN

```
**** SUBROUTINE DATA IN ******************************************
*      THIS SUBROUTINE  ALLOWS TYHE OPERATOR TO INPUT KEY BOARD *
*  DATA OR CHANGE THE DATA NEEDED FOR THE SECTION CALCULATION   *
*  AND DISPLAY.                                                 *
*****************************************************************
      SUBROUTINE DATA_IN

      INCLUDE 'TOP.FOR'
      INCLUDE 'TOP_MANU.FOR'

      STA=UIS$SET_POINTER_POSITION(VD_ID,WD_MANU,0.,10.2)
      KB_ID=UIS$CREATE_KB('SYS$WORKSTATION')
      CALL UIS$ENABLE_KB(KB_ID,WD_MANU)



      CALL DATA_LBL (VD_ID,DATA_UP)

1     CALL UIS$ERASE(VD_ID,-.4,10.1,3.4,10.9)
      CALL UIS$TEXT(VD_ID,7,'ENTER A LINE NUMBER',0.,11.)
      CALL UIS$TEXT(VD_ID,7,'OR [RETURN] TO EXIT',0.,10.6)
3     CALL  KEY_READ (LINE,.TRUE.,0.,12.,*300)
      READ (LINE,FMT='(F2.0)',ERR = 5) RLINE_NO
      LINE_NO = INT(RLINE_NO)
      GO TO (10,20,30,40,50,60,70,80,90,100,
     &    110,120,130,140,150,160,170,180,190,200) LINE_NO


*  IF IT GETS TO HERE A MISTAKE HAS BEEN MADE *******
5      CALL UIS$ERASE(VD_ID,-.4,10.1,3.4,10.9)
       CALL UIS$TEXT(VD_ID,7,'IMPROPER LINE NUMBER',0.,11.)
       CALL UIS$TEXT(VD_ID,7,'ENTER A NEW NUMBER PLEASE',0.,10.6)
       GOTO 3


10     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
       infile = line(:8)
       DATA_UP(1) = LINE
       CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
       GOTO 1

20     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)

       DATA_UP(2) = LINE
       READ (LINE, FMT='(G12.5)',ERR = 5) LENGTH
       CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)

**** CHECK TO SEE IF WE CAN CHANGE CB ? ****************
        IF ((BEAM .GT. 0.).AND.(DRAFT .GT. 0.).AND.(DSPMNT
```

```
     &       .GT. 0.)) THEN
             CB = DSPMNT * 35/(LENGTH*DRAFT*BEAM)
             WRITE (DATA_UP(6), FMT='(G12.5)') CB
             CALL UIS$ERASE(VD_ID,7.,15.8,9.9,16.2)
             CALL UIS$TEXT(VD_ID,7,DATA_UP(6),7.,16.2)
           END IF

           GOTO 1

30         CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
           DATA_UP(3) = LINE
           READ (LINE, FMT='(G12.5)',ERR = 5) BEAM
           CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)

           IF ((BEAM .GT.0.).AND.(DRAFT .GT. 0.) .AND.(DSPMNT
     &       .GT. 0.)) THEN
             CB = DSPMNT * 35/(LENGTH*DRAFT*BEAM)
             WRITE (DATA_UP(6), FMT='(G12.5)') CB
             CALL UIS$ERASE(VD_ID,7.,15.8,9.9,16.2)
             CALL UIS$TEXT(VD_ID,7,DATA_UP(6),7.,16.2)
           END IF
           GOTO 1

40         CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
           DATA_UP(4) = LINE
           READ (LINE, FMT='(G12.5)',ERR = 5) DRAFT
           CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)

           IF ((BEAM .GT.0.).AND.(LENGTH .GT. 0.) .AND.(DSPMNT
     &       .GT. 0.)) THEN
             CB = DSPMNT * 35/(LENGTH*DRAFT*BEAM)
             WRITE (DATA_UP(6), FMT='(G12.5)') CB
             CALL UIS$ERASE(VD_ID,7.,15.8,9.9,16.2)
             CALL UIS$TEXT(VD_ID,7,DATA_UP(6),7.,16.2)
           END IF
           GOTO 1

50         CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
           DATA_UP(5) = LINE
           READ (LINE, FMT='(G12.5)',ERR = 5) DSPMNT
           CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)

           IF ((BEAM .GT.0.).AND.(DRAFT .GT. 0.) .AND.(LENGTH
     &       .GT. 0.)) THEN
             CB = DSPMNT * 35/(LENGTH*DRAFT*BEAM)
             WRITE (DATA_UP(6), FMT='(G12.5)') CB
             CALL UIS$ERASE(VD_ID,7.,15.8,9.9,16.2)
             CALL UIS$TEXT(VD_ID,7,DATA_UP(6),7.,16.2)
           END IF
           GOTO 1

60         CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
           DATA_UP(6) = LINE
           READ (LINE, FMT='(G12.5)',ERR = 5) CB
           CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)

           IF ((BEAM .GT.0.).AND.(DRAFT .GT. 0.) .AND.(LENGTH
     &       .GT. 0.))THEN
             DSPMNT = CB * LENGTH * DRAFT * BEAM / 35
             WRITE (DATA_UP(5), FMT='(G12.5)') DSPMNT
```

```
            CALL UIS$ERASE(VD_ID,7.,16.2,9.9,16.6)
            CALL UIS$TEXT(VD_ID,7,DATA_UP(5),7.,16.6)
        END IF
        GOTO 1

70      CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(7) = LINE

        READ (LINE, FMT='(G12.5)',ERR = 5) XG
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

80      CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(8) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) TRIM
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

90      CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(9) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) TT
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

100     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(10) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) SPEED
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

110     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(11) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) AB
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

120     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(12) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) NRUD
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

130     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(13) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) ARUD
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

140     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(14) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) DEL_RUD
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

150     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(15) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) STERN
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1
```

```
160     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(16) = LINE
        READ (LINE, FMT='(G12.5)', ERR = 5) NPROP
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

170     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(17) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) DPTH
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

180     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(18) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) UNK
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

190     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(19) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) UNK
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

200     CALL FIND_IT(LINE_NO,DEL_Y,LINE,*300)
        DATA_UP(20) = LINE
        READ (LINE, FMT='(G12.5)',ERR = 5) UNK
        CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)
        GOTO 1

300     CALL UIS$DISABLE_KB(KB_ID)

        CALL UIS$ERASE(VD_ID,-.4,10.1,3.4,10.9)
        CALL UIS$TEXT(VD_ID,7,'SELECT AN OPTION',0.,11.)
        CALL UIS$TEXT(VD_ID,7,' WITH THE MOUSE ',0.,10.6)

        RETURN
        END
 ************************* END OF DATA_IN ***********************
```

APPENDIX I

SUBROUTINE FIND_IT

```
**** A ROUTINE FOR READING AND TRANSFER OF KEY STROKES****
*********************************************************
      SUBROUTINE FIND_IT(LINE_NO,DEL_Y,LINE,COUNT,*)

      INCLUDE 'TOP.FOR'

      REAL DEL_Y, DY
      LOGICAL FCHAR
      CHARACTER  LINE*12
      CHARACTER *25 MANU_INST1(20), MANU_INST2(20)

**** NOW FOR INSTRUCTIONS ***************************
      MANU_INST1(1) = 'ENTER A FILE NAME OF '
      MANU_INST2(1) = 'EIGHT CHARACTERS MAX '
      MANU_INST1(2) = 'ENTER THE SHIP LENGTH '
      MANU_INST2(2) = '     RETURN TO EXIT '
      MANU_INST1(3) = 'ENTER THE SHIPS BEAM '
      MANU_INST2(3) = '    RETURN TO EXIT'
      MANU_INST1(4) = 'ENTER THE SHIP DRAFT '
      MANU_INST2(4) = '    RETURN TO EXIT '
      MANU_INST1(5) = 'ENTER THE DISPLACEMENT '
      MANU_INST2(5) = ' CB IS RECALCULATED'
      MANU_INST1(6) = '     ENTER THE CB '
      MANU_INST2(6) = 'DISPLACEMENT RECOMPUTED'
      MANU_INST1(7) = 'ENTER THE LONG. COG. '
      MANU_INST2(7) = '    RETURN TO EXIT'
      MANU_INST1(8) = '  TRIAL TRIM IN FEET'
      MANU_INST2(8) = ' DOWN STERN = NEG.'
      MANU_INST1(9) = ' ENTER TRIAL DRAFT'
      MANU_INST2(9) = '    IN FEET  '
      MANU_INST1(10) = ' ENTER APPROACH SPEED'
      MANU_INST2(10) = '  IN KNOTS'
      MANU_INST1(11) = ' SUBMERGED PROFILE BOW'
      MANU_INST2(11) = ' AREA IN FEET**2'
      MANU_INST1(12) = ' ENTER NO. OF RUDDERS'
      MANU_INST2(12) = '    RETURN TO EXIT'
      MANU_INST1(13) = ' RUDDER AREA IN SQ.'
      MANU_INST2(13) = ' FEET OF ONE RUDDER'
      MANU_INST1(14) = ' ENTER THE RUDDER '
      MANU_INST2(14) = ' ANGLE IN DEGREES'
      MANU_INST1(15) = ' 1 FOR CLOSED STERN'
      MANU_INST2(15) = ' 2 FOR OPEN STERN'
      MANU_INST1(16) = ' NO. OF SCREWS'
      MANU_INST2(16) = '  RETURN TO EXIT'
      MANU_INST1(17) = ' WATER DEPTH IN FEET'
      MANU_INST2(17) = '    RETURN TO EXIT'
```

```
**** PRINT UP THE CORRECT INSTRUCTION *****************
       CALL UIS$ERASE(VD_ID,-.4,10.1,3.4,10.9)
       DEL_Y = 18.6 - .4 * LINE_NO
       CALL UIS$TEXT(VD_ID,7,MANU_INST1(LINE_NO),0.,11.)
       CALL UIS$TEXT(VD_ID,7,MANU_INST2(LINE_NO),0.,10.6)

**** DOES THE STRING NEED TO BE CONDITIONED? **********
       IF (LINE_NO .EQ. 1 ) THEN
          FCHAR = .FALSE.
       ELSE
        FCHAR = .TRUE.
       END IF

**** READ THE INPUT STRING AND WRITE IT UP ************
       CALL KEY_READ(LINE,FCHAR,0.,12.,*300)
       DY = DEL_Y - .35
       CALL UIS$ERASE(VD_ID,7.,DY,10.,DEL_Y)
       CALL UIS$TEXT(VD_ID,7,LINE,7.,DEL_Y)

       RETURN
300    RETURN 1
       END
*********************** END FIND_IT ******************
```

APPENDIX J

SUBROUTINE DATA_LBL

```
************************************************************************
*    A ROUTINE WHICH WRITES UP THE MANUEVERING DISPLAY LINE            *
*  LABLES.  NEED ONLY SUPPLY THE VD_ID NUMBER.   THIS MAY BE           *
*  CHANGED LATER TO WRITE IN PREVIOUS DATA.**************************

      SUBROUTINE DATA_LBL (VD_ID,DATA_UP)

      IMPLICIT INTEGER (A-Z)
      INCLUDE 'SYS$LIBRARY:UISENTRY'
      INCLUDE 'SYS$LIBRARY:UISUSRDEF'
      REAL DEL_Y
      CHARACTER *18 LINE_LBL(20), DATA_UP(20)*12

      CALL UIS$ERASE(VD_ID,4.,10.1,9.9,18.3)

**** STORE THE LINE LABELS FOR THE DISPLAY ************
      LINE_LBL(1) = '1. SHIP FILE NAME.'
      LINE_LBL(2) = '2. SHIP LENGTH ...'
      LINE_LBL(3) = '3. BEAM ..........'
      LINE_LBL(4) = '4. DRAFT .........'
      LINE_LBL(5) = '5. DISPLACEMENT...'
      LINE_LBL(6) = '6. CB ............'
      LINE_LBL(7) = '7. CENTER OF GRAV.'
      LINE_LBL(8) = '8. TRIM...........'
      LINE_LBL(9) = '9. TRIAL DRAFT....'
      LINE_LBL(10) = '10. SPEED .........'
      LINE_LBL(11) = '11. BOW AREA ......'
      LINE_LBL(12) = '12. N0. OF RUDDERS.'
      LINE_LBL(13) = '13. RUDDER AREA....'
      LINE_LBL(14) = '14. RUDDER ANGLE...'
      LINE_LBL(15) = '15. STERN TYPE.....'
      LINE_LBL(16) = '16. NO. OF SCREWS .'
      LINE_LBL(17) = '17. WATER DEPTH ...'
      LINE_LBL(18) = '18.   LATER  ......'
      LINE_LBL(19) = '19.   LATER  ......'
      LINE_LBL(20) = '20.   LATER  ......'
```

```
**** WRITE UP THE LABELS AND ANY DATA WHICH EXISTS ****
      DO 10 I = 1 ,20
         DEL_Y = 18.6 - .4*I
         CALL UIS$TEXT(VD_ID,7,LINE_LBL(I),4.5,DEL_Y)
         CALL UIS$TEXT(VD_ID,7,DATA_UP(I),7.,DEL_Y)
10    CONTINUE

      RETURN
      END
************************ END DATA_LBL *****************
```

APPENDIX K

SUBROUTINES TC_CAL AND TC_CAL_RPT

```
************************************************************
****** TC CAL CALCULATES THE STD, TD, AD, TR, AND VT **********
************************************************************
      SUBROUTINE TC_CAL

      INCLUDE 'TOP.FOR'
      INCLUDE 'TOP_MANU.FOR'

      CHARACTER TC_DATA(5)*12

**** CALL THE CALCUALTING ROUTINE *********************
      CALL TC_CAL_RPT

**** CONVERT THE NUMBERS TO CHARACTER STRINGS *********
      WRITE(TC_DATA(1), FMT='(G12.5)') STD
      WRITE(TC_DATA(2), FMT='(G12.5)') TD
      WRITE(TC_DATA(3), FMT='(G12.5)') AD
      WRITE(TC_DATA(4), FMT='(G12.5)') TR
      WRITE(TC_DATA(5), FMT='(G12.5)') VT

      CALL UIS$ERASE(VD_ID,7.5,10.6,9.9,12.6)

**** DISPLAY THE NEW DATA **************  ************
      DO 20 I = 1,5
         DEL_Y = 13. - I * .4
         CALL UIS$TEXT(VD_ID,7,TC_DATA(I),7.6,DEL_Y)
20    CONTINUE

      RETURN
      END
************************** END TC_CAL ************************


************************************  ***********************
**  THIS ROUTINE DOES THE TURNING CIRCLE CALCULATIONS.  IT     *
* IS SPLIT OUT IN ORDER TO BE CALLABLE FROM THE REPORT SECTION *
************************************************************

      SUBROUTINE TC_CAL_RPT

      INCLUDE 'TOP.FOR'
      INCLUDE 'TOP_MANU.FOR'

      IF (NPROP .LT. 1.1) THEN
         STD = (4.19 +(194. - 203.*CB)/ABS(DEL_RUD) + .7*(TT-DRAFT - 1)
     2         *(DEL_RUD/ABS(DEL_RUD))*(STERN - 1.)) * TRIM**2
     2         + 47.4*TRIM - 13.0*BEAM + ((3.82*(STERN-1.)
     2         -35.8*(STERN-1.))*ARUD + 7.79*AE)/TT
         TD = .91*STD + .424*SPEED*LENGTH**.5 + .675*LENGTH
         AD = .519*TD + 1.33*LENGTH
         TR = .497*TD - .065*LENGTH
```

```
            VT = (.074*TD/LENGTH + .149)*SPEED

         ELSE

            STD = (.727+(188-197*CB)/ABS(DEL_RUD))*LENGTH
        2    +4.65*BEAAAAM+41*TRIM
        2    +(25.56*AB-218*ARUD*(NRUD-1))/TT
        2    +3.2*SPEED*LENGTH**.5

            TD = STD+.14*LENGTH
            AD = .514*TD+1.1*LENGTH
            TR = .531*TD-.357*LENGTH
            VT = (.028*TD/LENGTH+.543)*SPEED

         ENDIF

         RETURN
         END
*************************** END TC_CAL_RPT ****************
```

APPENDIX L

SUBROUTINE RECORDIT

```
************************************************************************
*    SUBROUTINE RECORDIT SAVES THE INPUT DATA WITH FILE NAME           *
* OF SHIPNAME AND FILE TYPE .DAT AUTOMATICALLY.                        *
************************************************************************
      SUBROUTINE RECORDIT

      INCLUDE 'TOP.FOR'
      INCLUDE 'TOP_MANU.FOR'
      LOGICAL *1 USER
      REAL DUMMY
      CHARACTER   FNAME*12
      STA = UIS$SET_POINTER_POSITION(VD_ID,WD_MANU,0.,10.)

**** OPEN A NEW FILE AND WRITE THE DATA TO IT **************
      OPEN(2,FILE = INFILE, STATUS = 'NEW',blank='null')
      REWIND (2)
      WRITE (2,100) INFILE
      WRITE (2,110) LENGTH, BEAM, DRAFT, DSPMNT
      WRITE (2,110) CB, XG, TRIM,TT
      WRITE (2,110) SPEED, AB, NRUD, ARUD
      WRITE (2,110) DEL_RUD, STERN, NPROP, DPTH

**** AND WRITE THE HYD. DERIVATIVES INTO THE FILE **********
      WRITE (2,FMT ='(I4)') USER
      WRITE (2,110) YVD,YRD,NVD,NRD
      WRITE (2,110) YV,YR,NV,NR
      WRITE (2,110) YD,ND,DUMMY,DUMMY
      ENDFILE (2)
      CLOSE(2)

**** AND REPORT OUT WHEN DONE ******************************
      FNAME = INFILE(:INDEX(INFILE,' '))//'.DAT'
      CALL UIS$ERASE(VD_ID,-.4,10.1,3.4,10.9)
      CALL UIS$TEXT(VD_ID,7,'FILE SAVED AS',0.,11.)
      CALL UIS$TEXT(VD_ID,7,FNAME,0.,10.6)

100   FORMAT (BN,A)
110   FORMAT (4G12.5)
      RETURN
      END
************************ END RECORDIT ************************
```

APPENDIX M

SUBROUTINE READ_FILE

```
*****************************************************************
*     READ FILE IS A SUBROUTINE WHICH READS THE DATA INPUT FILE  *
*  SPECIFIED BY KEYBOARD ENTRY.                                  *
*****************************************************************

       SUBROUTINE READ_FILE

       INCLUDE 'TOP.FOR'
       INCLUDE 'TOP_MANU.FOR'
       STA=UIS$SET_POINTER_POSITION(VD_ID,WD_MANU,0.,10.2)

**** CREATE AND ENABLE THE KEYBOARD ************************
       KB_ID=UIS$CREATE_KB('SYS$WORKSTATION')
       CALL UIS$ENABLE_KB(KB_ID,WD_MANU)

**** WRITE UP THE INSTRUCTION ******************************
       CALL UIS$ERASE(VD_ID,-.4,10.1,2.3,10.9)
       CALL UIS$TEXT(VD_ID,7,'ENTER THE FILE NAME',0.,11.)
       CALL UIS$TEXT(VD_ID,7,'AND FILE EXTENSION',0.,10.6)

**** READ THE FILE AS FLOATING POINT NUMBERS ***************
       CALL KEY_READ(LINE,'TRUE',0.,12.,*150)
       OPEN(3,FILE = LINE, STATUS = 'OLD')
       REWIND (3)
       READ (3,100) INFILE
       READ (3,110) LENGTH,BEAM, DRAFT,DSPMNT
       READ (3,110) CB, XG, TRIM, TT
       READ (3,110) SPEED, AB, NRUD, ARUD
       READ (3,110)  DEL_RUD, STERN, NPROP, DPTH
       REWIND 3

**** NOW READ IT IN AS CHARACTER DATA TO PRINT TO SCREEN ****
       READ (3, 100) DATA_UP(1)
         DO 10 I = 2, 14,4
            I1 = I + 1
            I2 = I + 2
            I3 = I + 3
            READ (3,120) DATA_UP(I),DATA_UP(I1),DATA_UP(I2),DATA_UP(I3)
10       CONTINUE

100    FORMAT (A8)
110    FORMAT (4G12.5)
120    FORMAT (4A)

150    CLOSE(3)


**** WRITE UP THE DATA *************************************
       CALL DATA_LBL(VD_ID,DATA_UP)
       CALL UIS$DISABLE_KB(KB_ID)
       CALL UIS$ERASE(VD_ID,-.4,10.1,3.4,10.9)
```

```
      CALL UIS$TEXT(VD_ID,7,'SELECT AN OPTION',0.,11.)
      CALL UIS$TEXT(VD_ID,7,' WITH THE MOUSE ',0.,10.6)
      RETURN
      END
************************ END READ_FILE **********************
```

REFERENCES

1. Fanney, Richard H., Lt/USN, AE 4900 Final Project for Computer Graphics in Survivability, AE 4900 Report, Naval Postgraduate School, Monterey, California, June 1989.

2. Gillmer, Thomas C., and Johnson, Bruce., Introduction to Naval Architecture. Naval Institute Press, 1982.

3. Parsons, Michael G., Informal Course Notes For NA470 Ship Design II, Department of Naval Architecture and Marine Engineering, University of Michigan, Ann Arbor, December 1985.

4. Digital Equipment Corporation,MicroVMS Programmers Manual, April 1986.

5. PRINCIPLES OF NAVAL ARCHITECTURE, v.1, Society of Naval Architects and Marine Engineers, 1988.

6. Clasrke, D., Gedling, P., and Hine, G., " The Application of Manoeuvering Criteria in Hull Design Using Linear Theory", The Royal Institute of Naval Architects, pp. 45-68, March, 1983.

7. Lyster, C. A. and Knights, H. L., " Prediction Equations for Ships' Turning Circles", Northeast Coast Institution of Engineers and Shipbuilders Transactions, v.93, pp.217-229. 23 April 1979.

8. Digital Equipment Corporation, MicroVMS Graphics Programming Guide, May 1986.

9. Nomoto, K., Taguchi, T., Honda, K., and Hirano, S., "On the Steering Qualities of Ships", International Shipbuilding Progress, v.4, no. 35, July 1957.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3. Department Chairman, Code 69 Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4. Naval Engineering Curricular Office, Code 34 Naval Postgraduate School Monterey, California 93943-5000	1
5. Professor Fotis A. Papoulias, Code 69Pa Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	3
6. Professor Young S. Shin, Code 69Sg Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	1
7. Professor James F. Hallock, Code 69Hi Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	1
8. Professor Robert E. Ball, Code AA/Bp Department of Aeronautical Engineering Naval Postgraduate School Monterey, California 93943-5000	1
9. Gerald K. McGowan, Lt/USN 1102 Spruance Road Monterey, California 93940	2